Preparing CMS for the HL-LHC and the future of computing

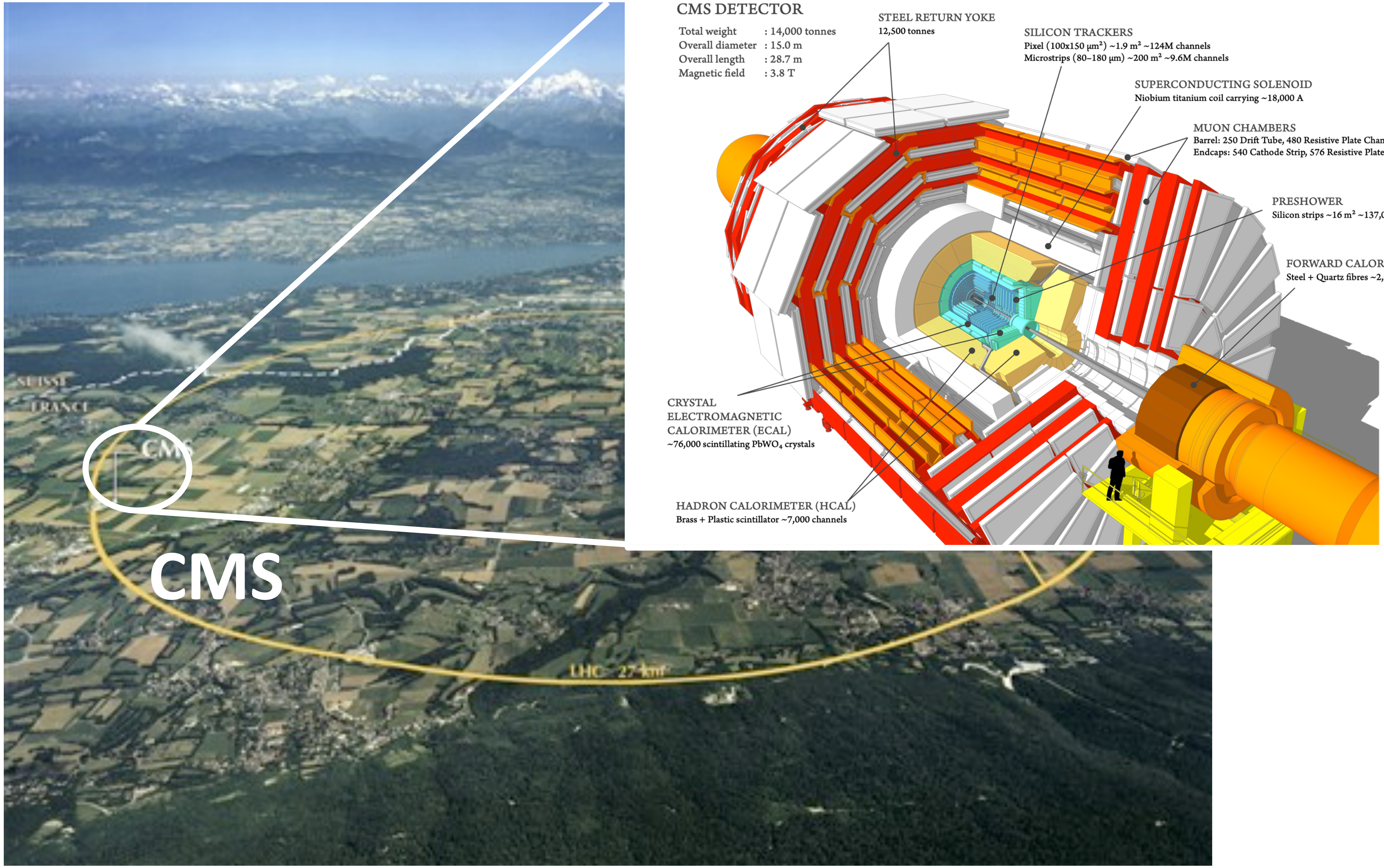
Allison Reinsvold Hall
Research Associate, Fermilab

BNL Particle Physics Seminar

Outline

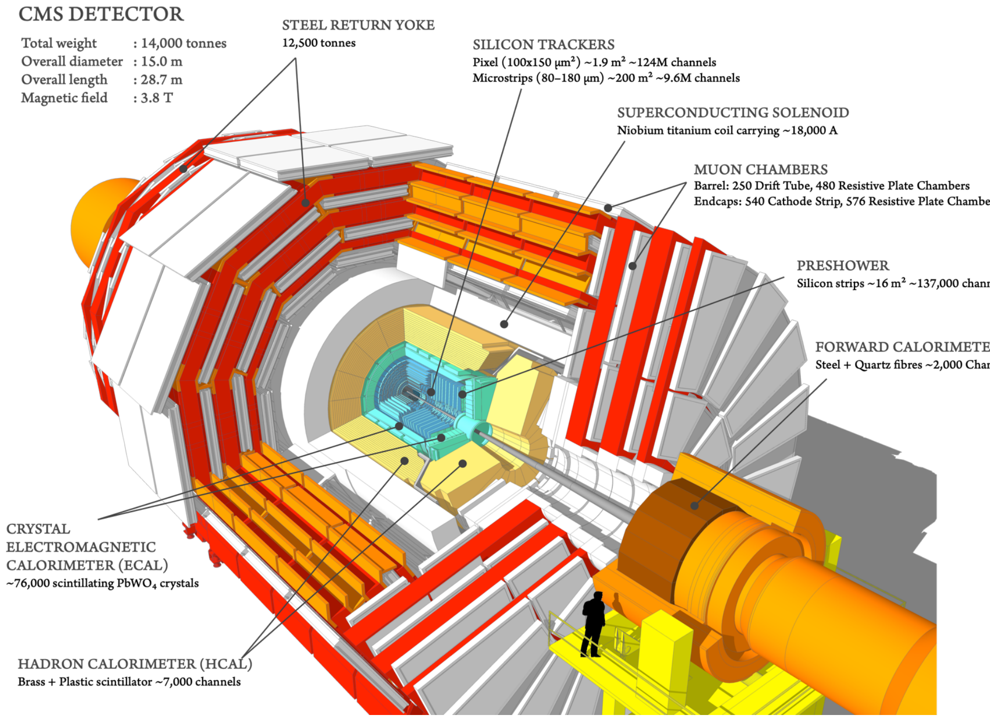
LHC and HL-LHC computing challenges → Reconstruction software: mkFit parallel tracking → Analysis software: Columnar analysis with Coffea

Compact Muon Solenoid (CMS)



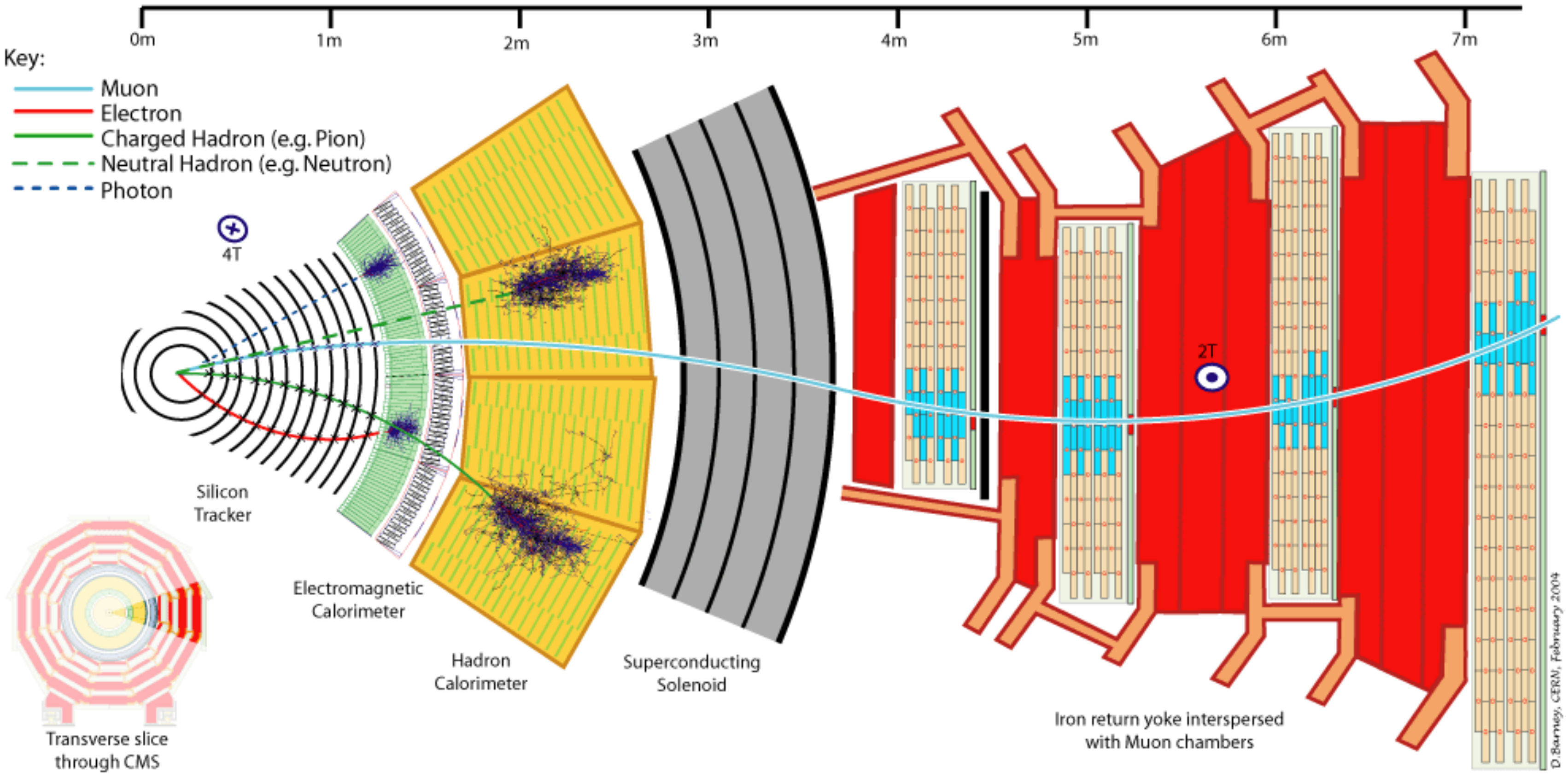
CMS computing

- 135 M detector channels
- 200 PB of data and simulation from Run 2
- Worldwide computing grid of >200k CPU cores
- 75 billion events processed per year
  - Monte Carlo simulation
  - (Re)processing data events



Reconstruction algorithms

Reconstruction: process of identifying particles and their properties (p_T, η, φ, etc) by their signatures in the different subdetectors of CMS



1. Reconstruct signatures **in each subdetector**

   Examples: tracks, calorimeter clusters

2. Reconstruct particles using the **complete event information**

   Example: Reconstruct muons from a track in the silicon tracker and hits in the outer muon system

Analysis workflow

Centrally produced data sets of recorded and simulated events
- Several tiers, each with reduced content
- RECO (Mb/ev) → AOD (500 kb/ev) → MiniAOD (50 kb/ev)

Ntupling (on grid)
Producing slimmed ROOT files with only the variables needed for your specific analysis

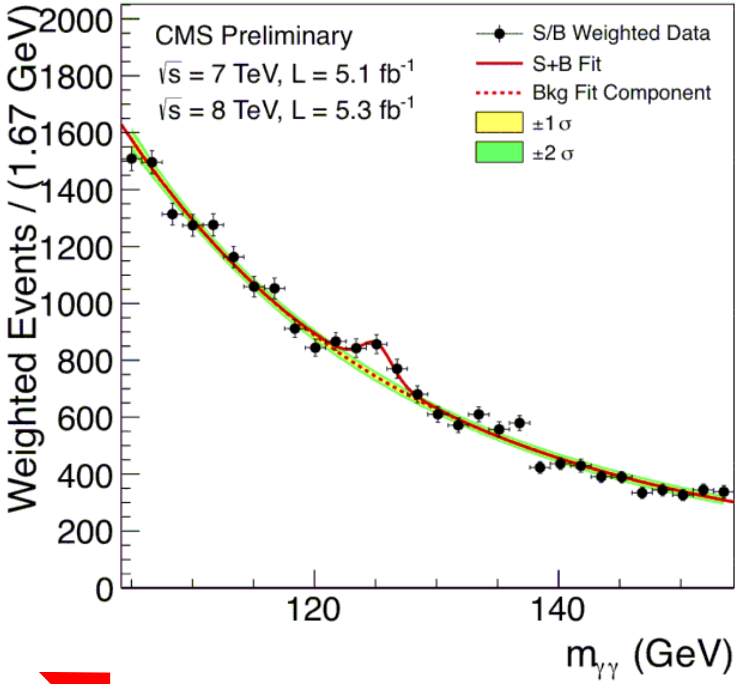
1–2 weeks, few times per year

Group ntuples or centrally produced **NanoAOD** (5 kb/ev)

Analysis code (locally or in batch)
Define signal and control regions, apply scale factors and corrections, estimate backgrounds, perform statistical analysis

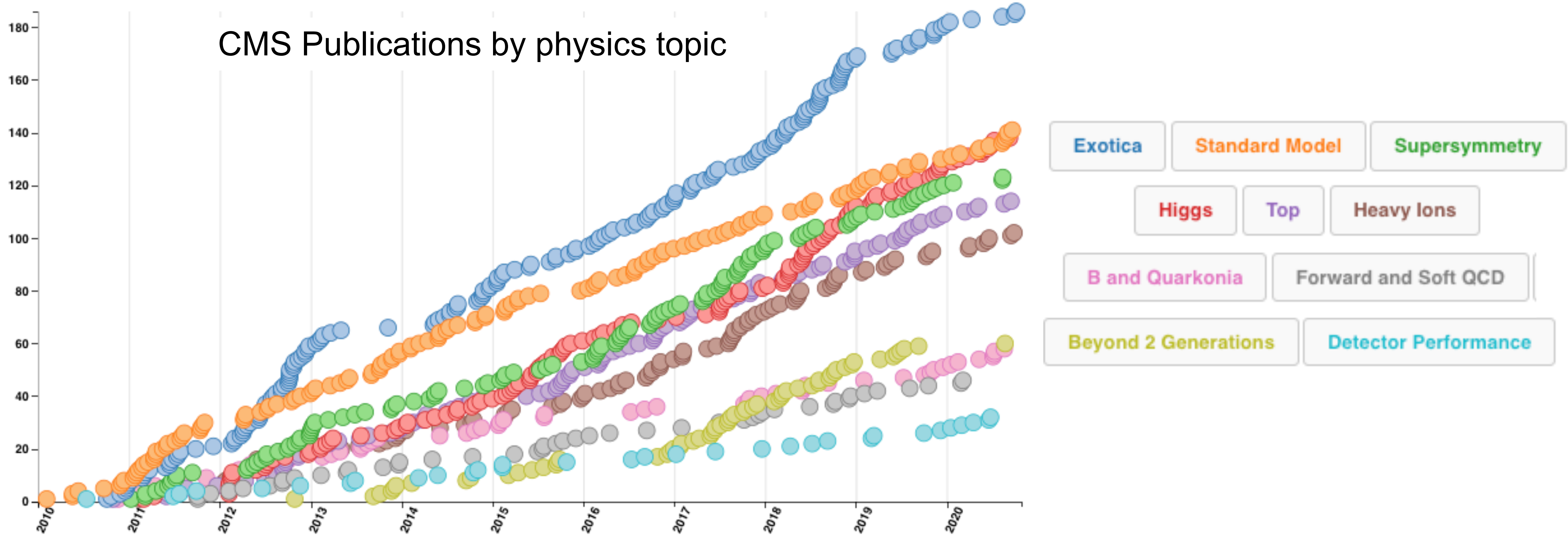
Several times per day

Final plots and tables



Physics results

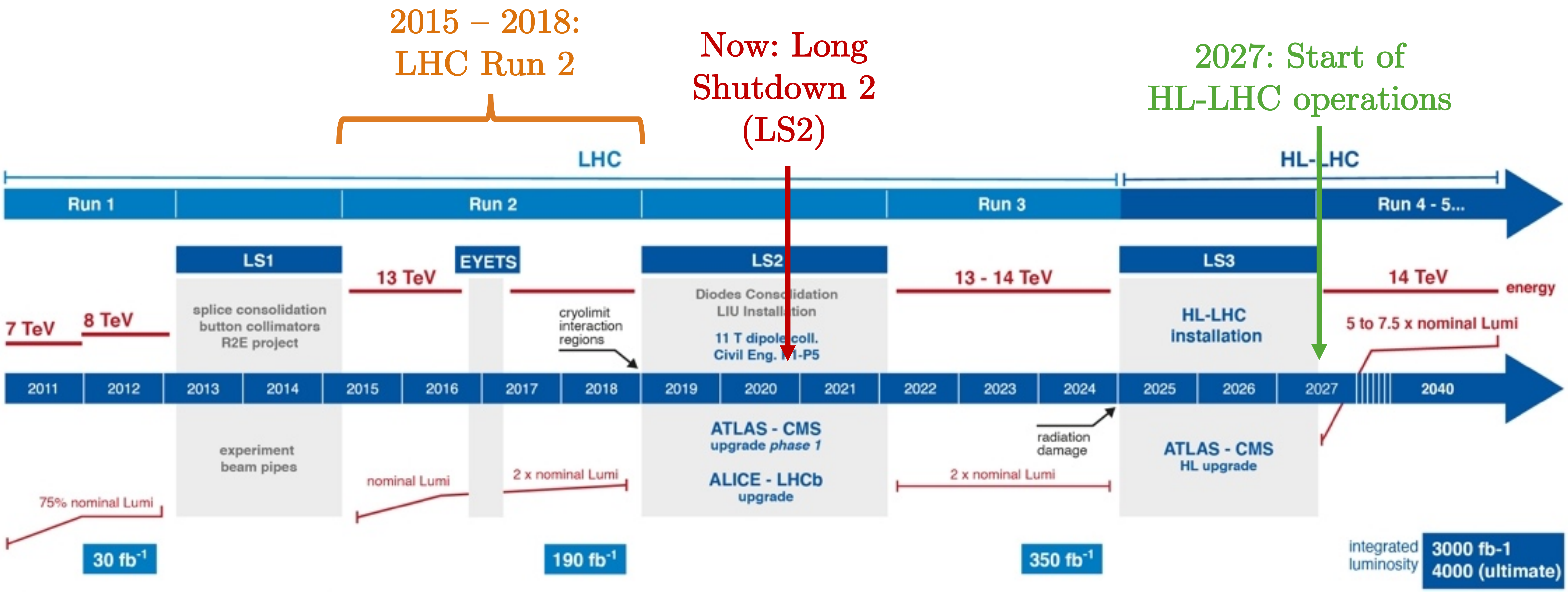
CMS recently published its 1000th physics paper!



LHC Timeline

- Integrated luminosity $\mathcal{L} = 160\ \text{fb}^{-1}$ in Run 2; expected to reach $\mathcal{L} > 3000\ \text{fb}^{-1}$ during High-Luminosity LHC (HL-LHC)

  $\rightarrow$ Many exciting physics opportunities ahead!



HL-LHC computing

Good news: more physics!

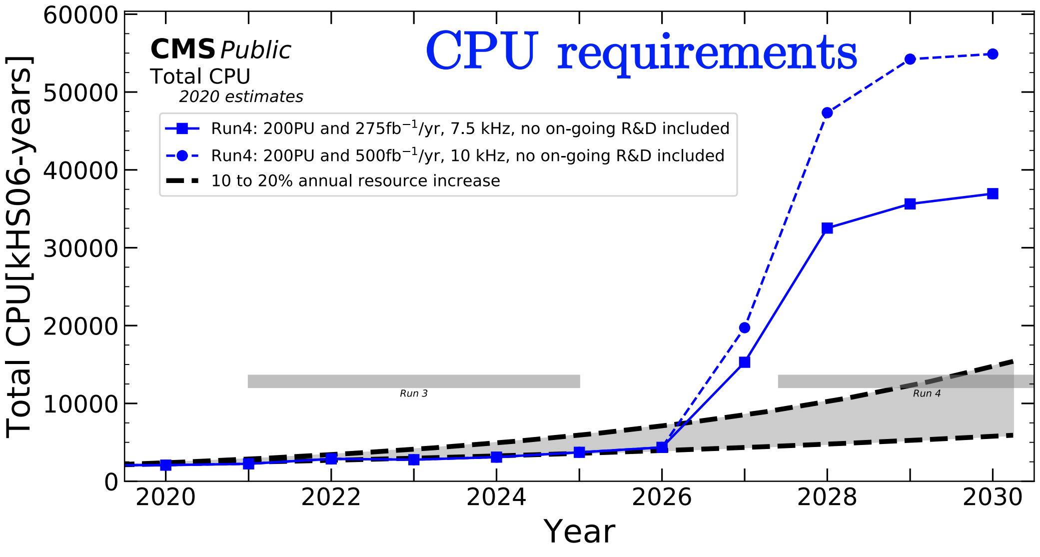
- HL-LHC will allow us to probe **even rarer** processes and **improve precision** on standard model measurements

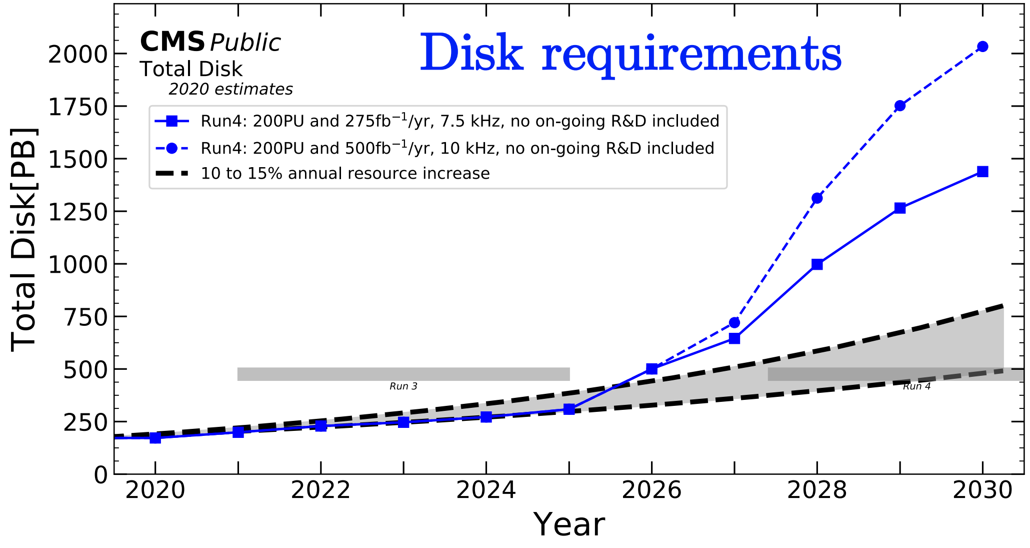
Big challenges: computing

- Instantaneous luminosity (collisions per second) will go up by > 5x
- Simultaneous overlapping proton-proton collisions (**pileup, PU**) will increase from 40 in Run 2 to 200 in the HL-LHC

→ **Need substantial computing R&D to enable HL-LHC physics**

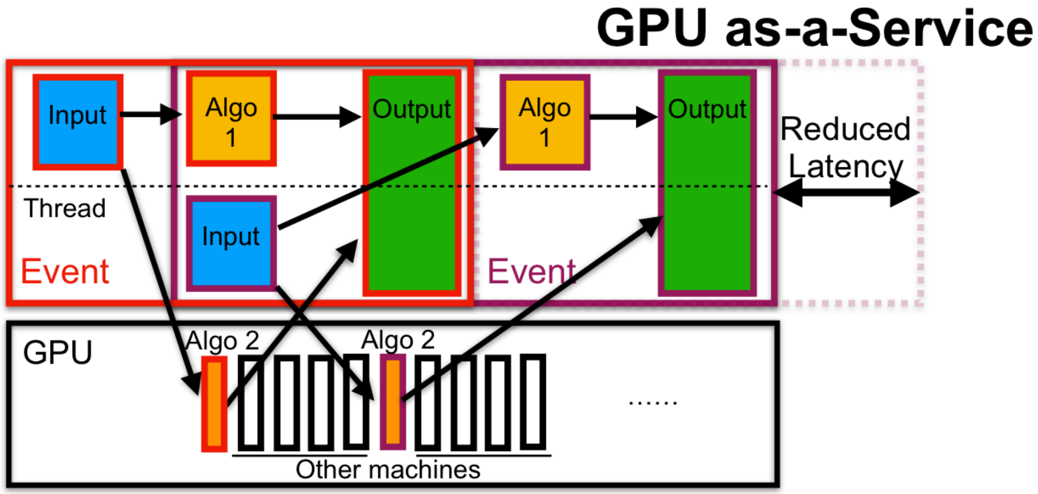
Assuming no R&D:





Computing R&D

- Heterogenous computing
  - GPUs will likely be used in the CMS software trigger during Run 3
    - CUDA versions of pixel tracking, local calorimeter reco. written by Patatrack team[1]
  - SONIC project[2] is exploring using GPUs "as-a-service", with multiple CPUs making calls to a GPU on an independent server
- Machine learning
  - Increased use in all aspects of CMS computing
  - Being explored for CMS high-granularity calorimeter (HGCAL) reconstruction[3] or track reconstruction (Exa.Trkx project[4])
- High performance computing (HPC) and cloud resources
- Parallelization and optimization
  - Use tools from data science industry
  - Optimize existing algorithms to take advantage of parallelization



1. Patatrack [2008.13461]
2. SONIC [2007.10359] (diagram above)
3. HGCAL reconstruction [ICHEP talk]
4. Exa.Trk.X [https://exatrkx.github.io/]

Parallelization methods

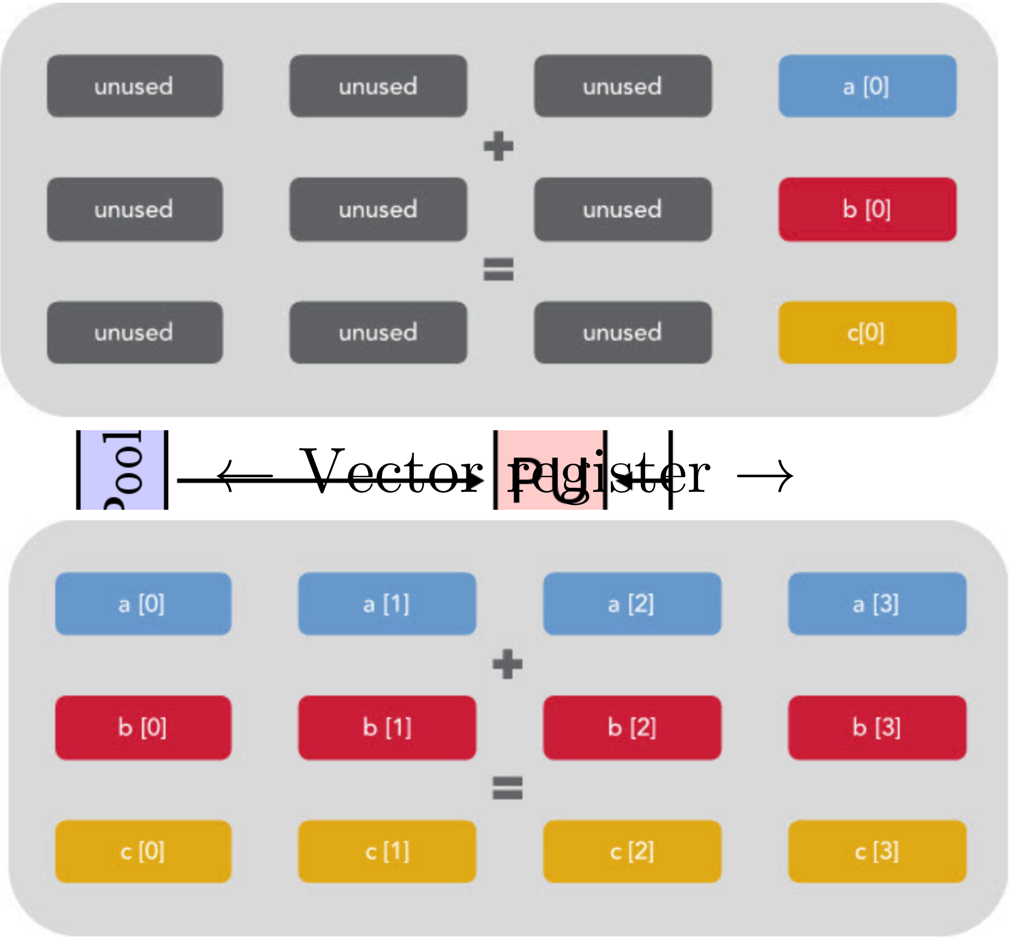
Two different forms of parallel code:

Multithreading

- Perform different tasks at the same time on different pieces of data
- Utilizes different CPU cores or hyperthreads

Vectorization:

- SIMD operations = Single-Instruction Multiple-Data
- Perform the same operation at the same time in lock-step across different data
- Utilizes vector registers on the CPU

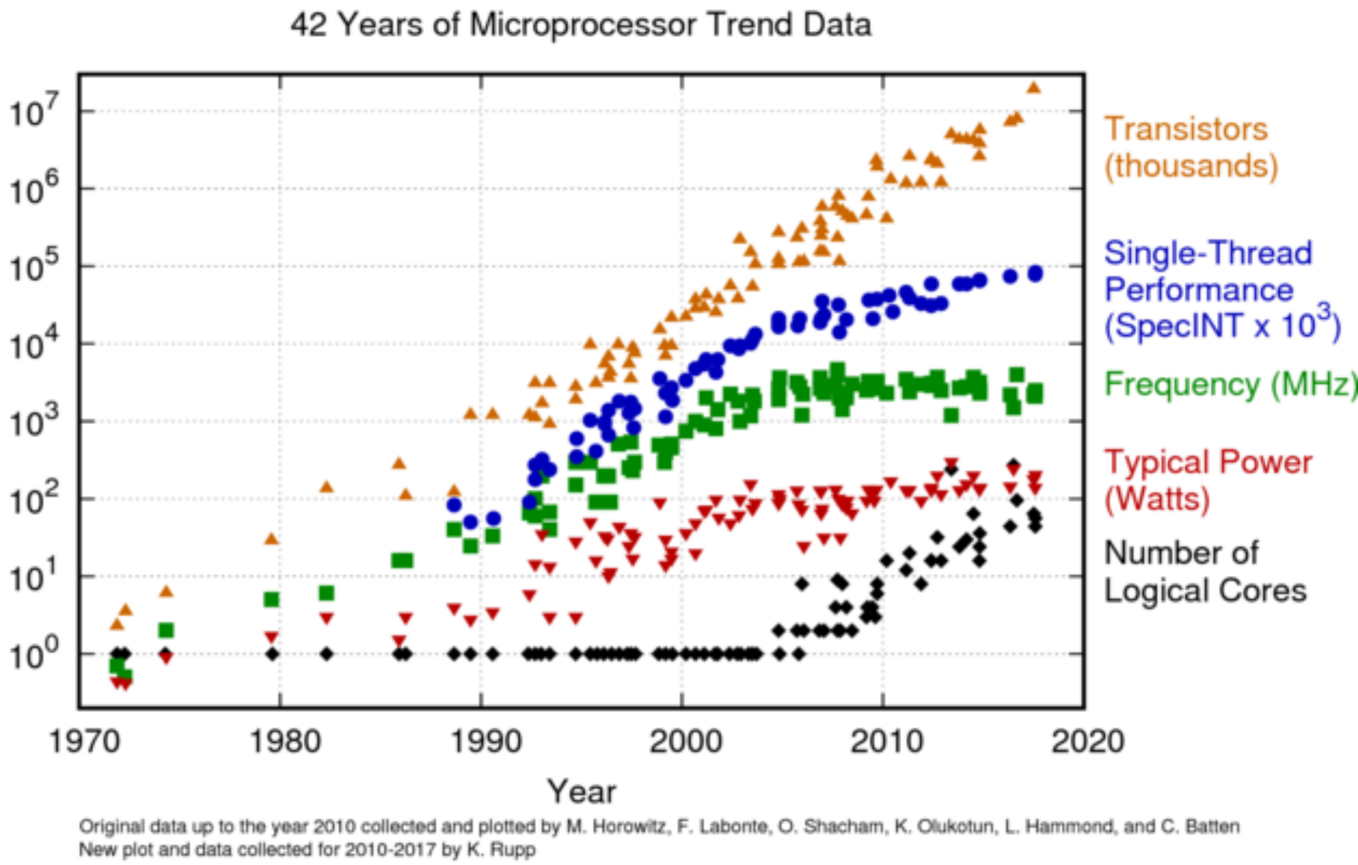


4x faster

* Showing one CPU clock cycle = shortest time slice it takes to perform a single operation

Computing trends

- Can no longer rely on frequency (CPU clock speed) to keep growing exponentially — nothing for free anymore
- Since 2005, most of the gains in single-thread performance come from vector operations
- But, **number of logical cores** is rapidly growing
- Rewrite algorithms to take advantage of both **multithreading** and vectorization



Outline

LHC and HL-LHC computing challenges

CMS computing

HL-LHC timeline

Parallelization methods

Reconstruction software: mkFit parallel tracking

Analysis software: Columnar analysis with Coffea

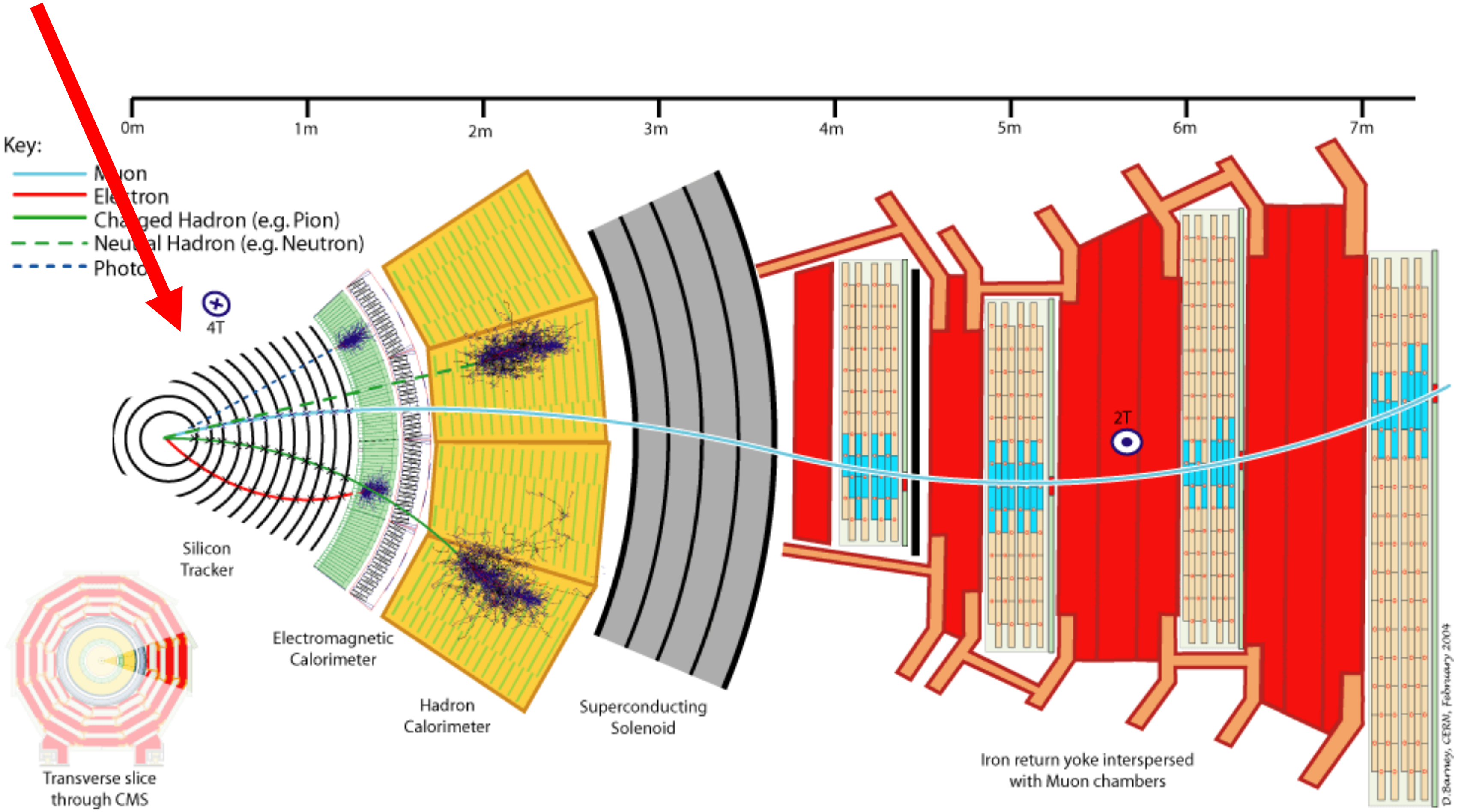
Speeding up reconstruction software: mkFit parallel track building

JINST 15 (2020) 09, arXiv:2006.00071

Project website: http://trackreco.github.io/

Reconstruction algorithms

Reconstruction: process of identifying particles and their properties (p_T, η, φ, etc) by their signatures in the different subdetectors of CMS



1. Reconstruct signatures **in each subdetector**

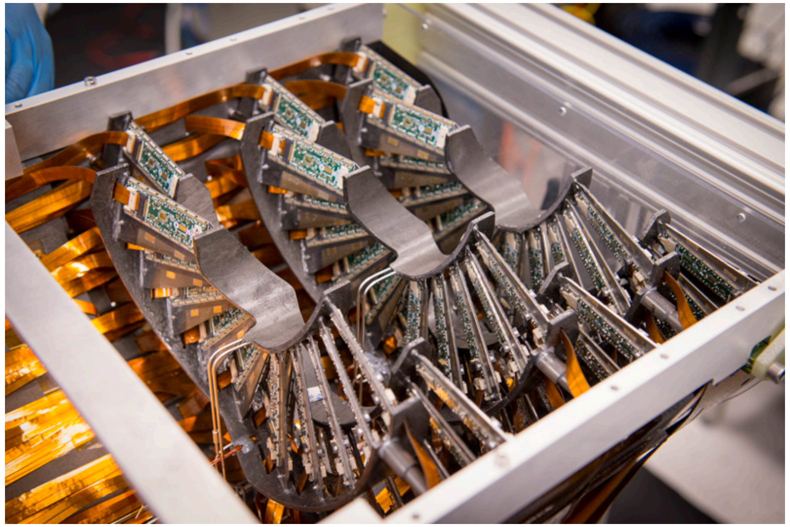
   Examples: tracks, calorimeter clusters

2. Reconstruct particles using the **complete event information**

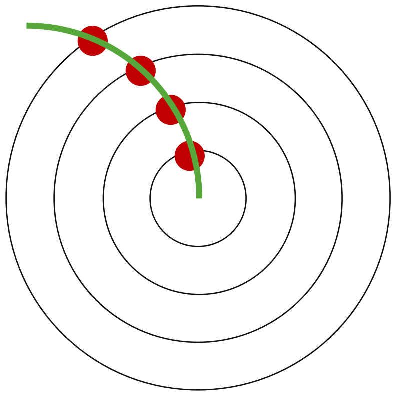
   Example: Reconstruct muons from a track in the silicon tracker and hits in the outer muon system

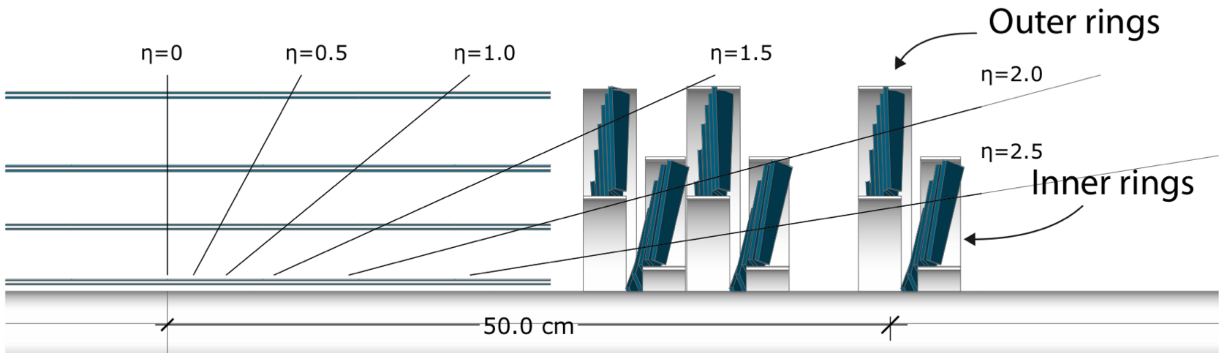
Tracker

- Closest detectors to the beamline
  1. Silicon pixel detector: pixel size 100μm x 150μm, 124M channels
  2. Silicon strip detector: strips are 80-200μm wide, 10M channels
- "Tracking" is the process of reconstructing charged particle trajectories from **hits** in the detector



Half endcap disks for the upgraded CMS pixel detector, installed early 2017

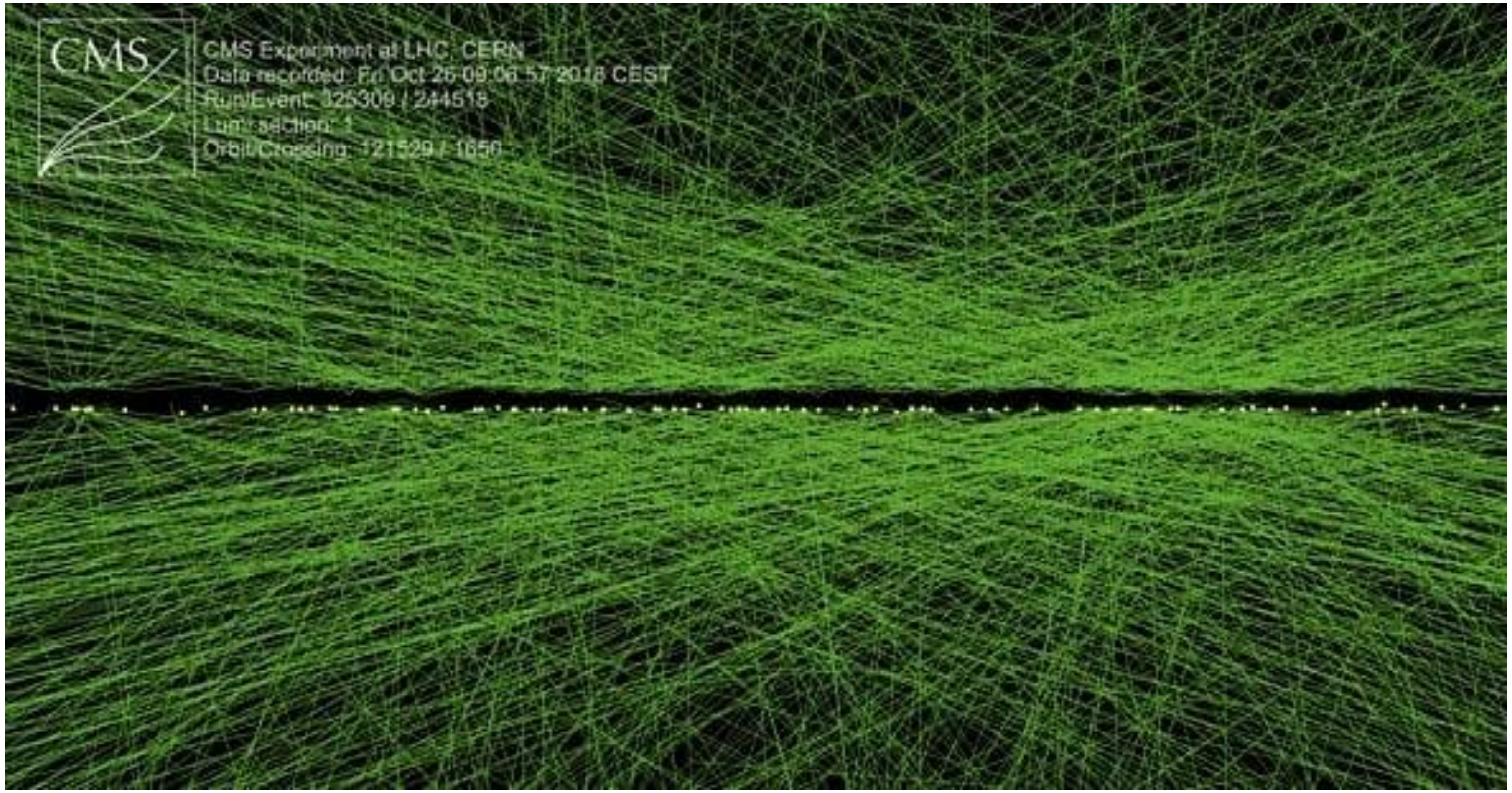




Layout of pixel detector, after 2017 upgrade

Setting the stage

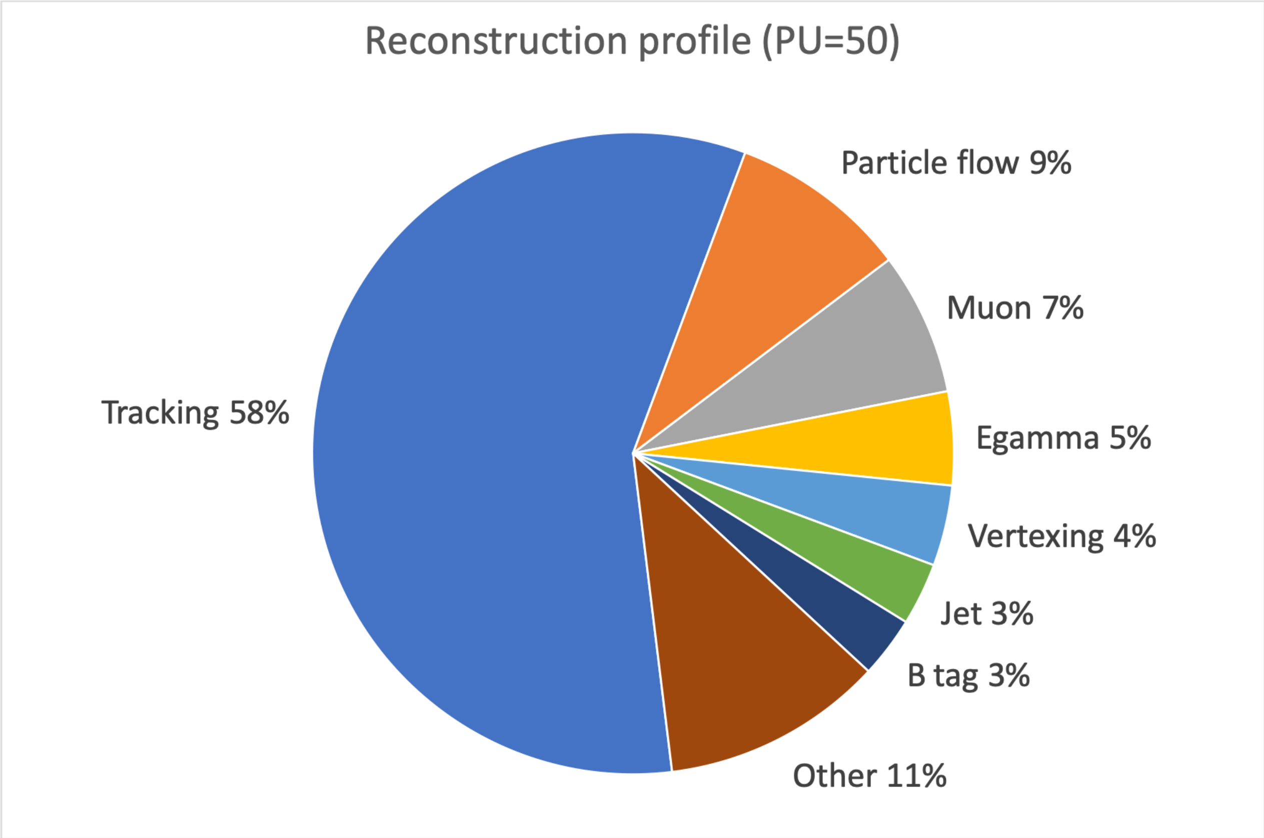
1. **Tracking is crucial:** for b-quark tagging, p_T^{miss}, PU mitigation, long-lived particles searches...



Event display, CMS 2018 high PU run (PU 136)

Setting the stage

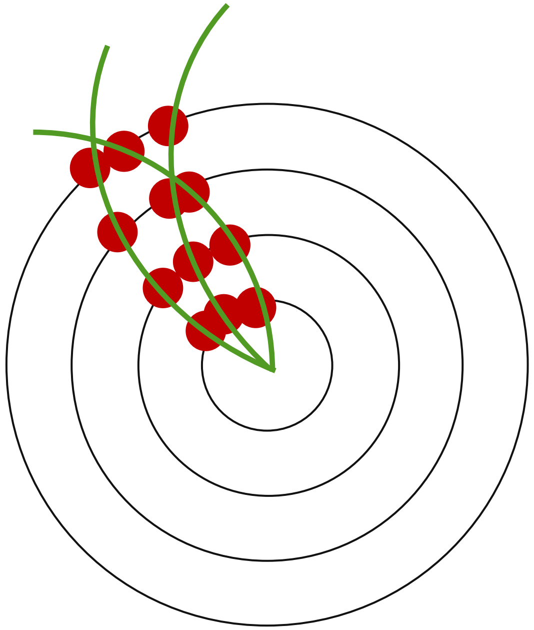
1. Tracking is crucial
2. **Tracking is time-consuming**

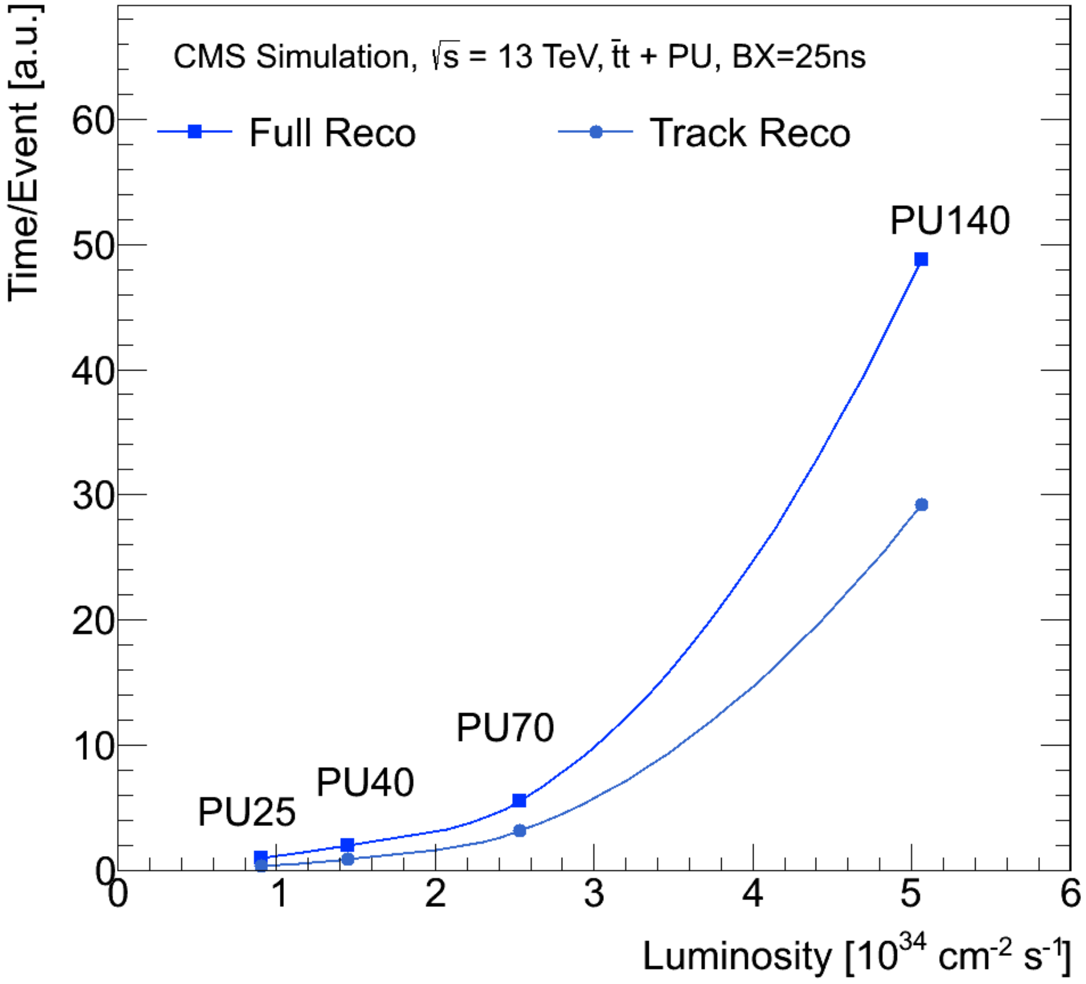


Profile of reconstruction time in CMS Software framework, CMSSW

Setting the stage

1. Tracking is crucial
2. Tracking is time-consuming
3. **Tracking times goes up dramatically with increased pileup**
   - Problem of combinatorics as occupancy increases



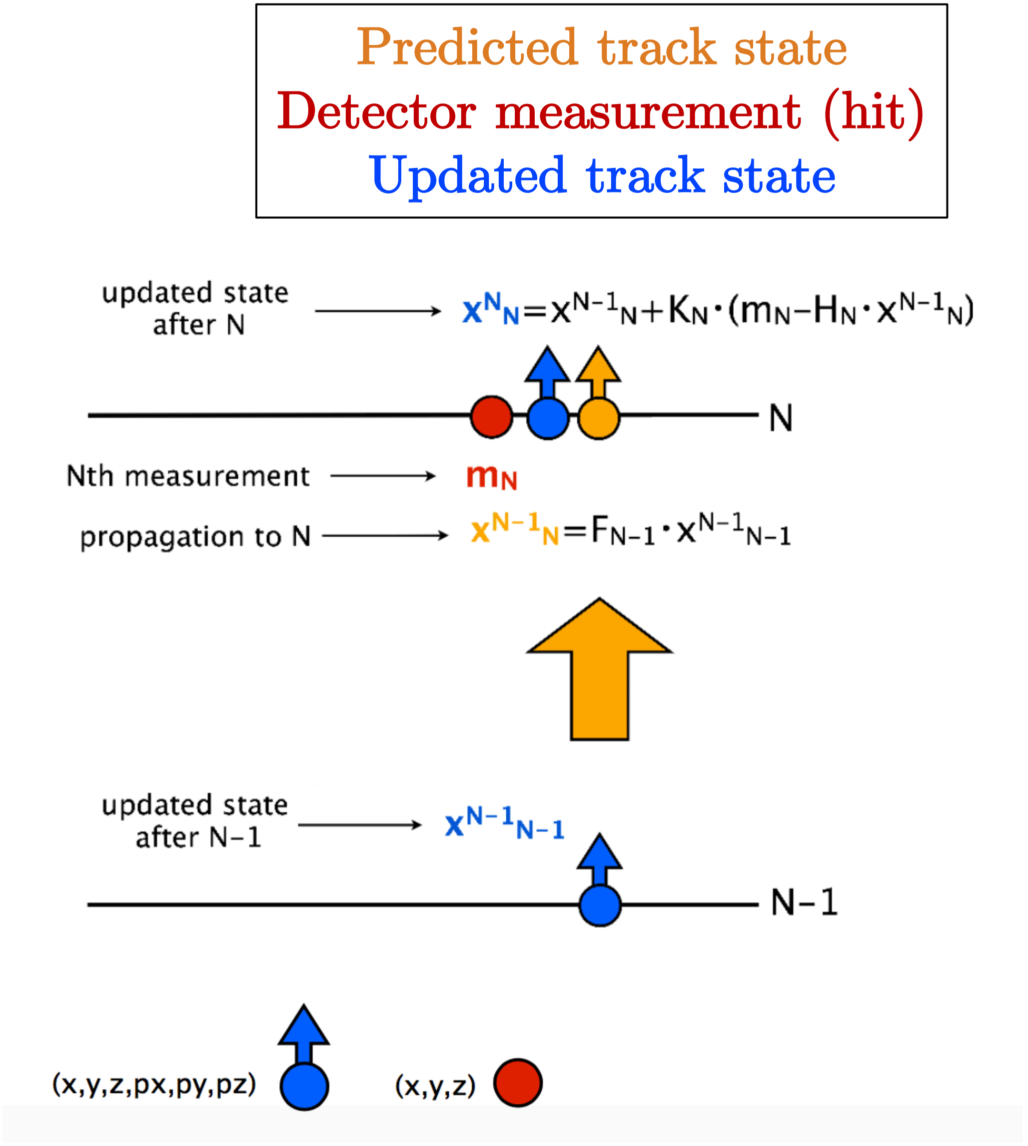


Kalman filter (KF) track building

- CMS uses a Kalman Filter algorithm for tracking
  - Demonstrated physics performance
  - Robust handling of multiple scattering, energy loss, and other material effects

Three step process:

1. **Propagate** the track state from layer N-1 to layer N (prediction)
2. **Search** for compatible hits on layer N
3. **Update** the track state using the hit parameters



Kalman Filter Performance

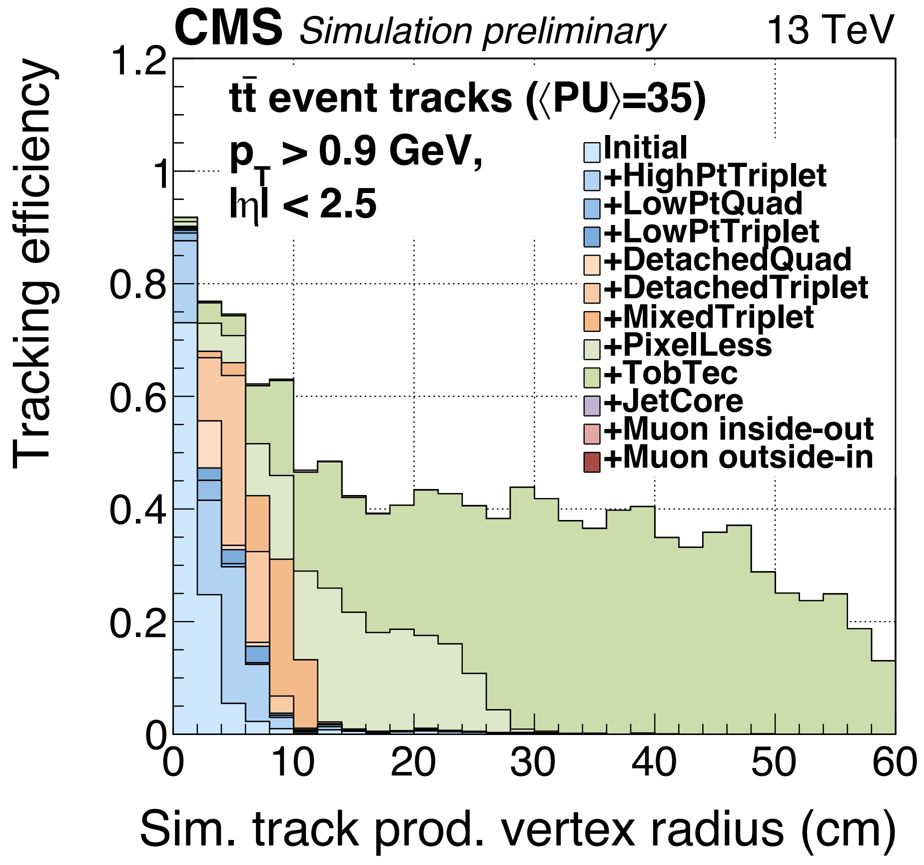
- Current CMS algorithm achieves excellent efficiency using KF track building
- Iterative approach:
  - Start with easiest tracks to build, remove associated hits, then look for more difficult tracks
  - Reduces combinatorics for later iterations

<u>**mkFit project:**</u>

A. Maintain excellent physics performance

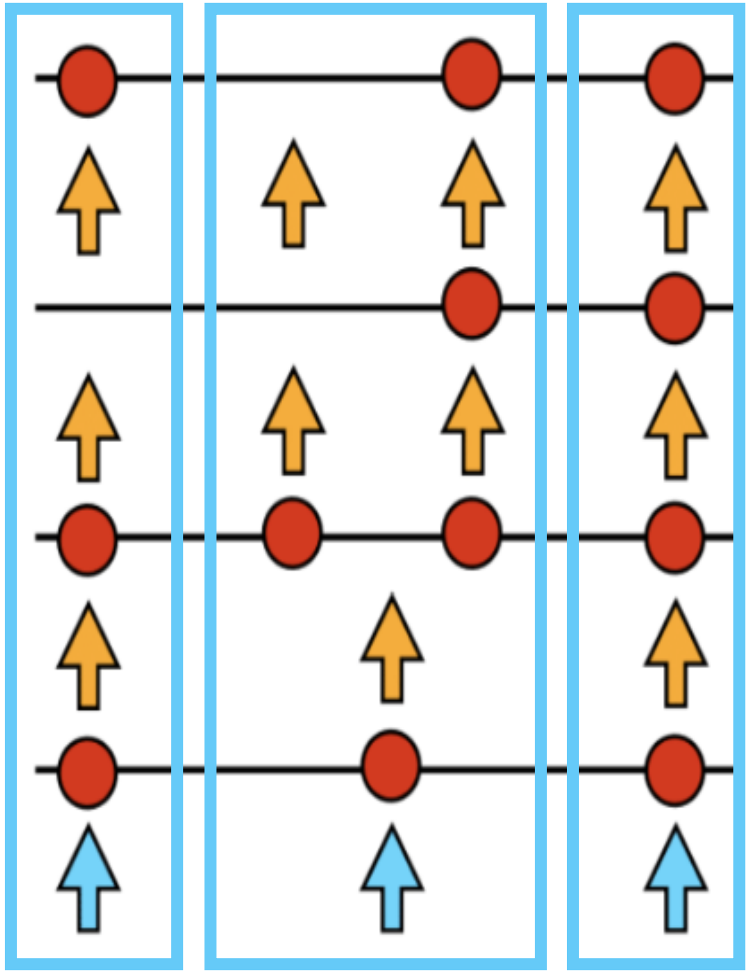
B. Speed up track building by taking advantage of parallel architectures

- Focus on initial iteration, which is responsible for building most prompt (ie, not displaced) tracks
- Aim for deployment in CMSSW in Run 3



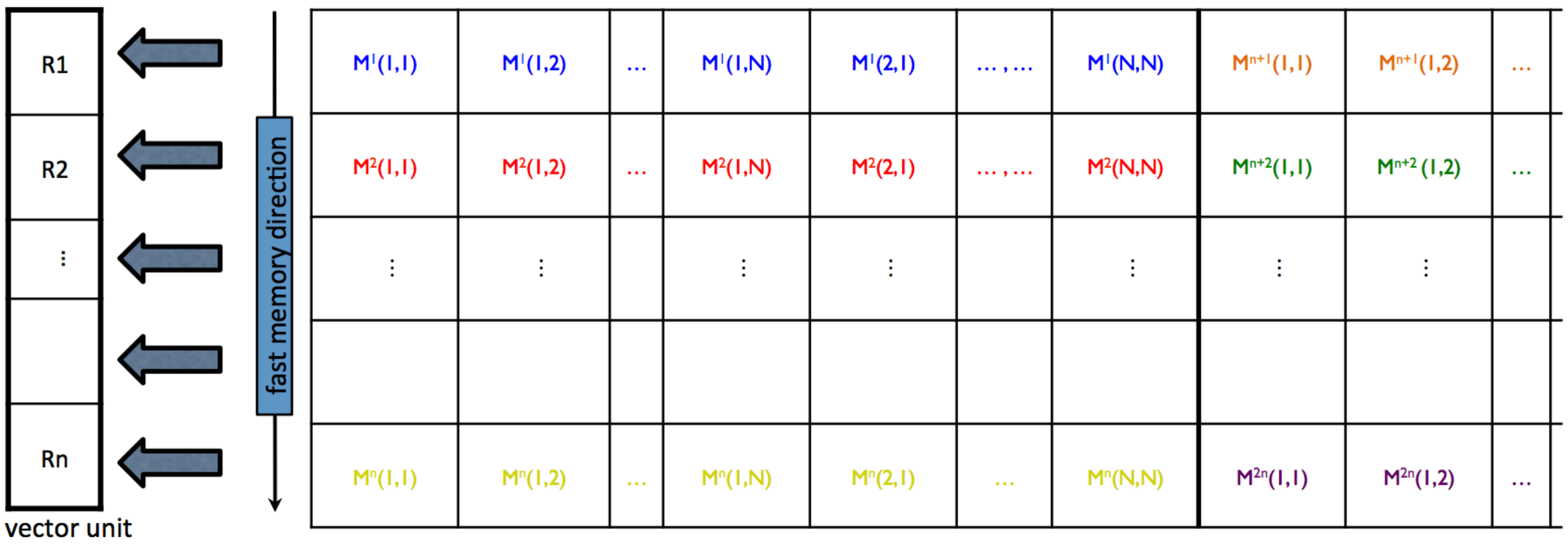
Challenges of KF track building

- KF track building is **not straightforward to parallelize**
- In track building, don't know which hits belong to which track
  - Start with a seed track
  - On each layer, could find 0, 1, 2+ hits compatible with the track
- Vectorization is hard: Requires **branching** to explore many possible track candidates
- Multithreading is hard: tracks differ in number of hits and events differ in number of tracks



How we do it

- **Multithreading** at nested levels using TBB
  - **parallel for:** N events in flight
    - **parallel for:** 5 regions in η in each event
      - **parallel for:** batches of 16 or 32 seeds per batch
- **Vectorized** processing of individual track candidates using both compiler vectorization and the custom-built Matriplex library



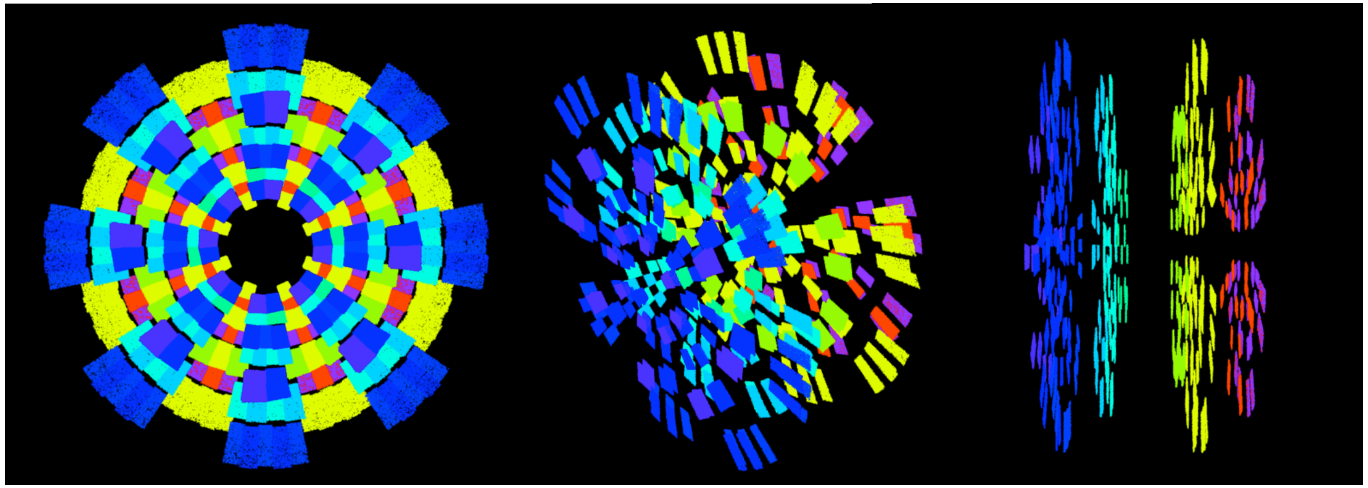
Matrix size **NxN**, vector unit size **n**

Matriplex

- Fill each vector unit with the same element from **n** different matrices and operate on each matrix in sync
- For example, the 6x6 covariance matrices for each track

Geometry in mkFit

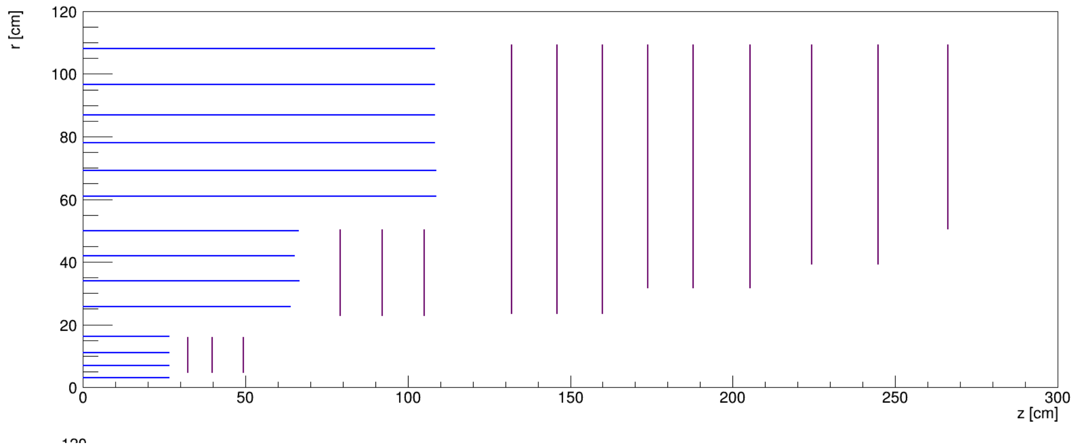
- Unlike CMSSW, **choose not** to deal with detector modules, only layers
  - Makes algorithm faster, more lightweight
- Geometry implemented as a plugin: core algorithm is **entirely separate** from detector geometry



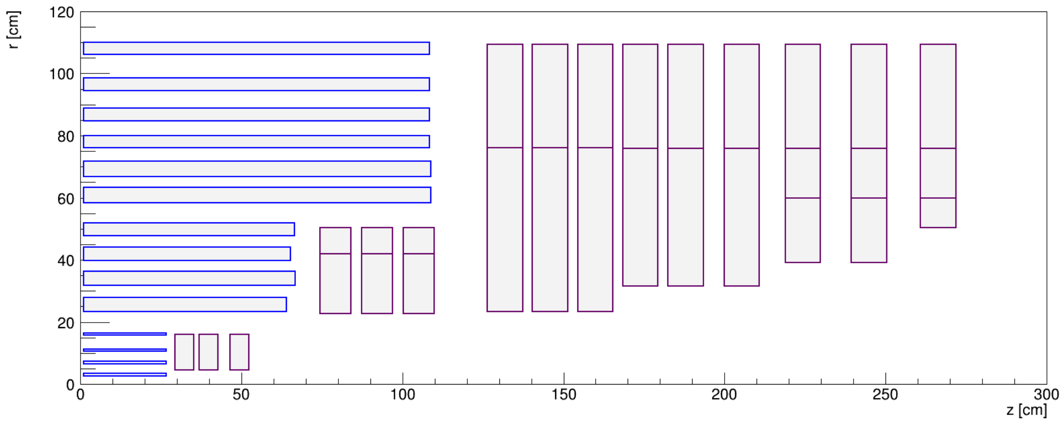
CMS endcap disk

Actual geometry used by MkFit

Layer centroids

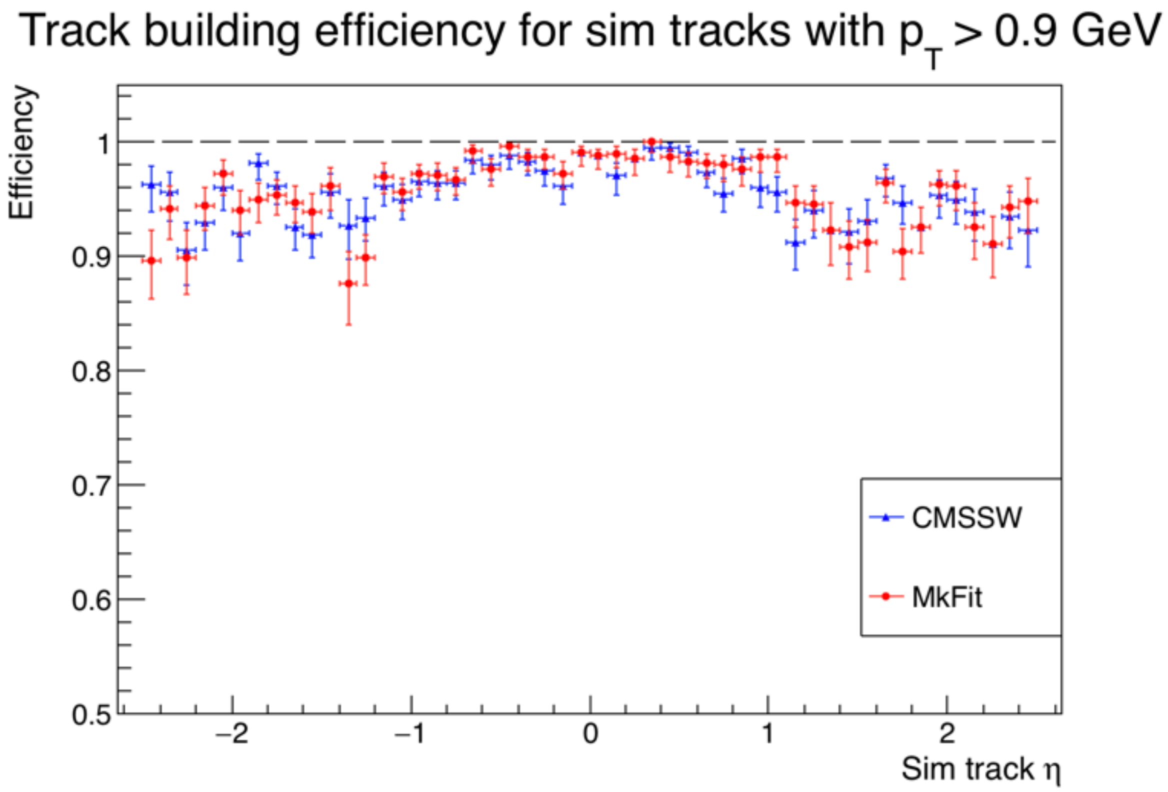


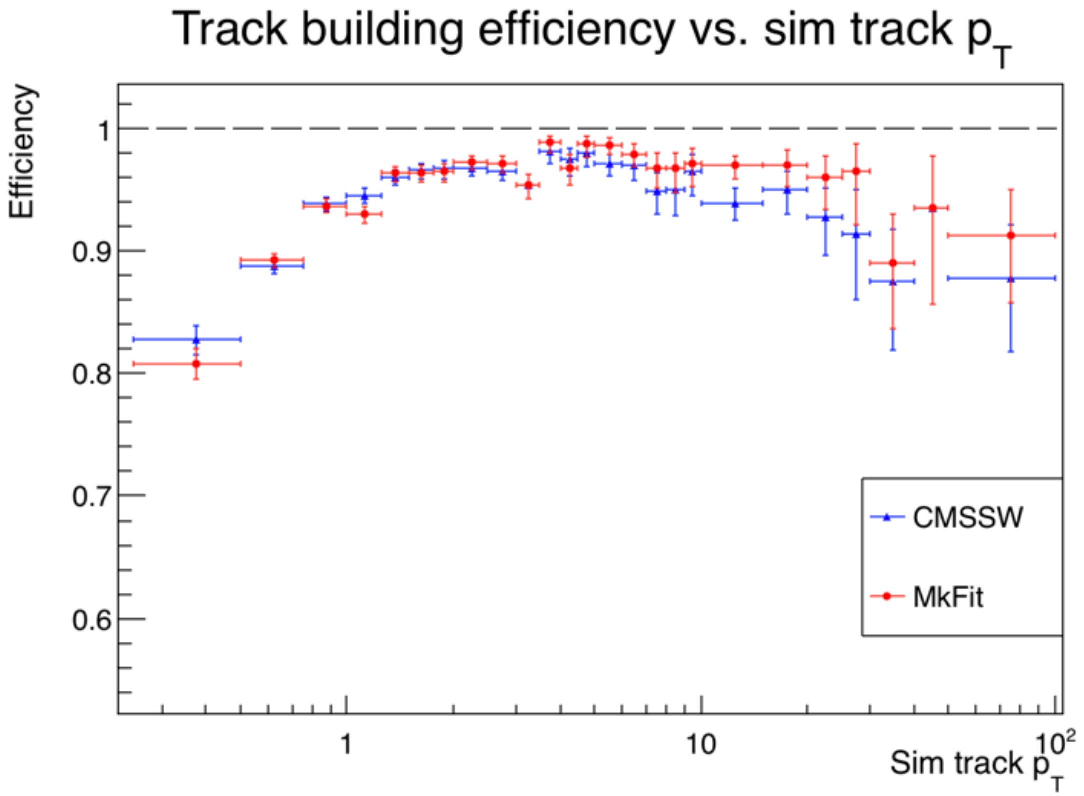
Layer size



Efficiency

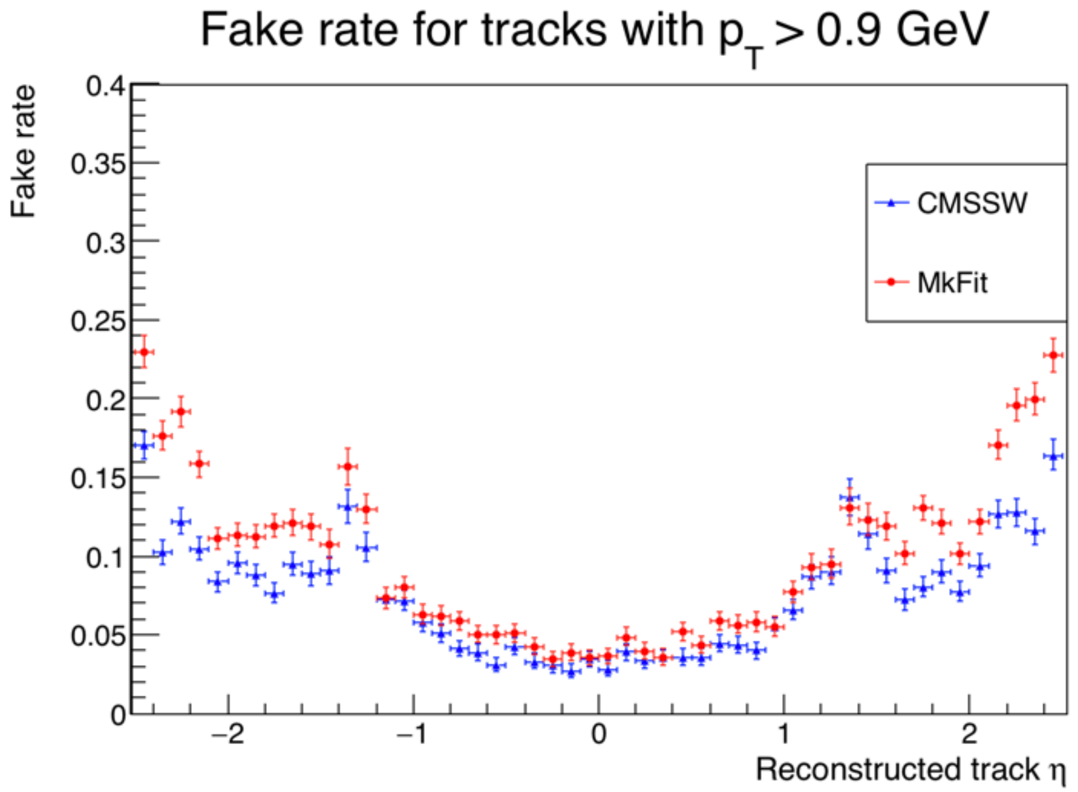
- **Efficiency:** fraction of simulated tracks that are matched to a reconstructed track
- **mkFit** is at least as efficient as **current CMS algorithm** across p_T and η
- Tested on $t\bar{t}$ events with average pileup of 50, CMS tracker geometry from 2018

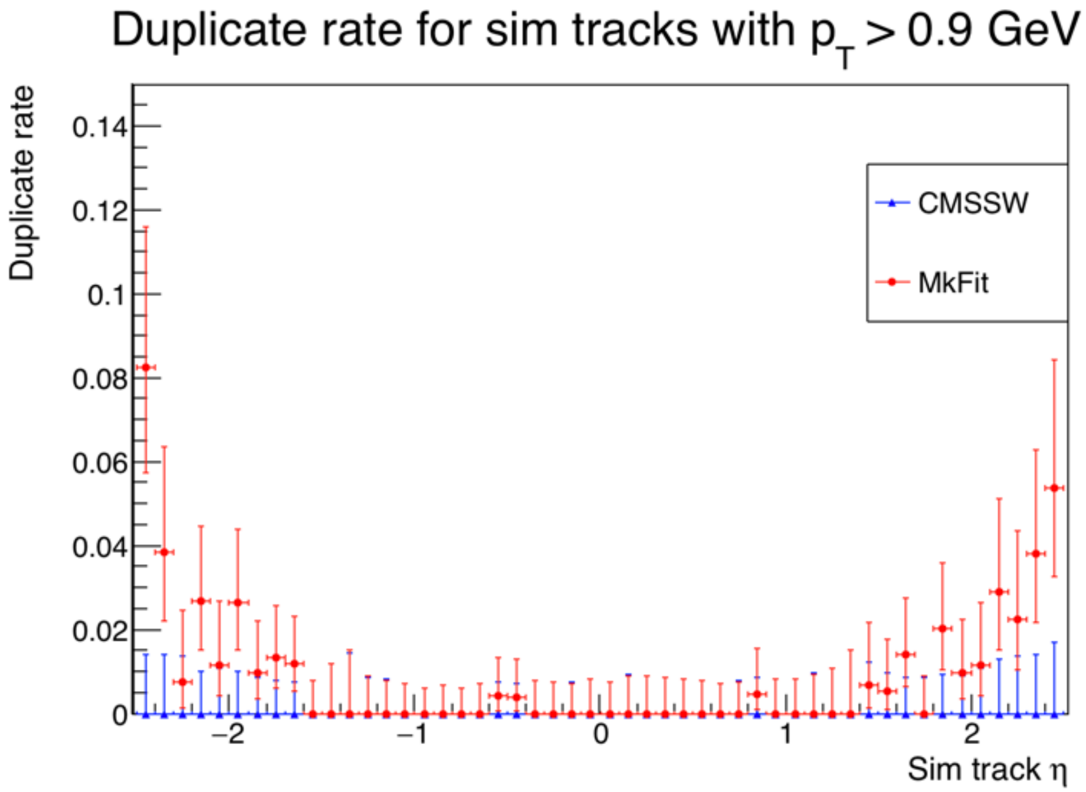




Fake rate and duplicate rate

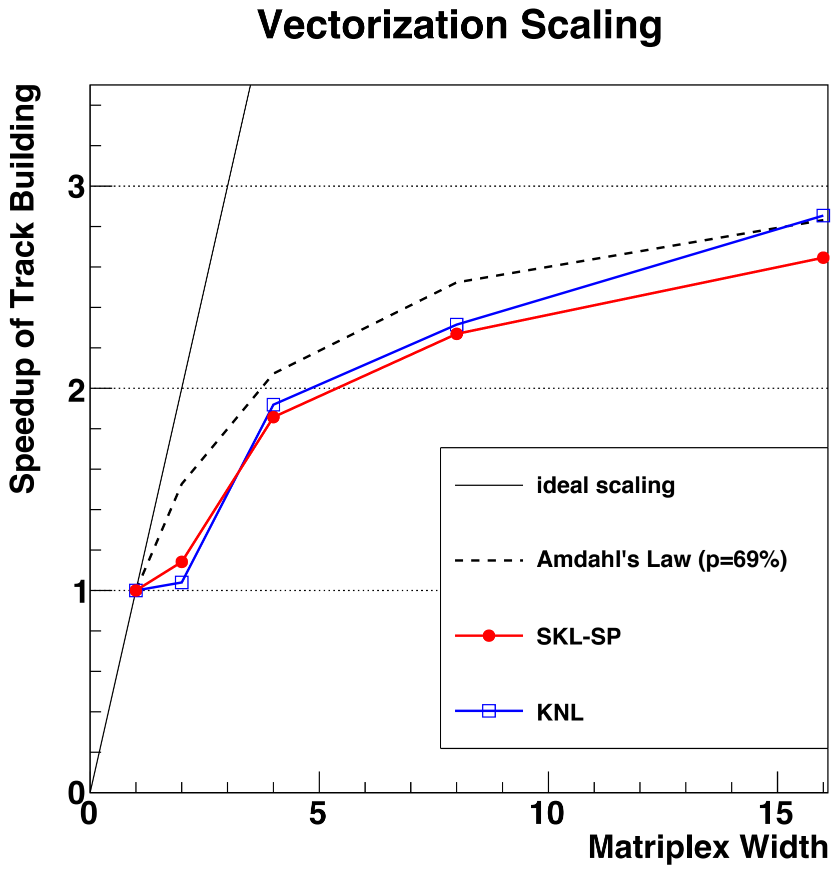
- **Fake rate:** fraction of reco tracks that are **not** matched to a sim track
- **Duplicate rate:** fraction of sim tracks that are matched to >1 reco track
- Small but manageable increase in fake rate and duplicate rate
  - Further optimizations ongoing





Vectorization performance

- Vectorization performance measured by artificially restricting the Matriplex width (how many matrices we calculate simultaneously)
  - Indicates close to 70% of code is successfully vectorized



Amdahl's Law

$$S = \frac{1}{(1-p) + {}^{p}/_{R}}$$

S = measured speedup

p = parallelized fraction

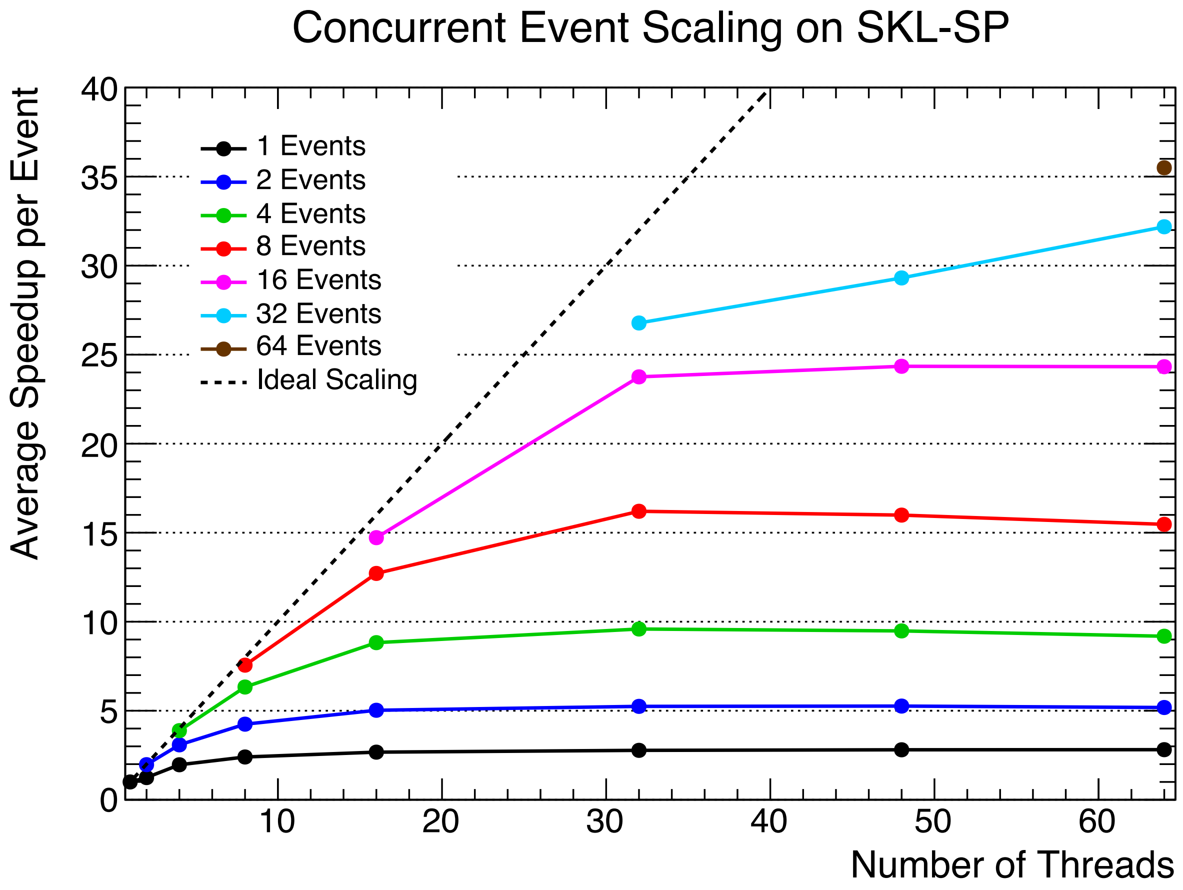
(1 – p) = remaining serial fraction

R = ratio of available to original resources (here, Matriplex width)

* mkFit run on a single-thread, track building times only

Multithreading performance

- Processing multiple events at a time allows the latencies between events to be hidden
- Maximum speedup of 35x achieved

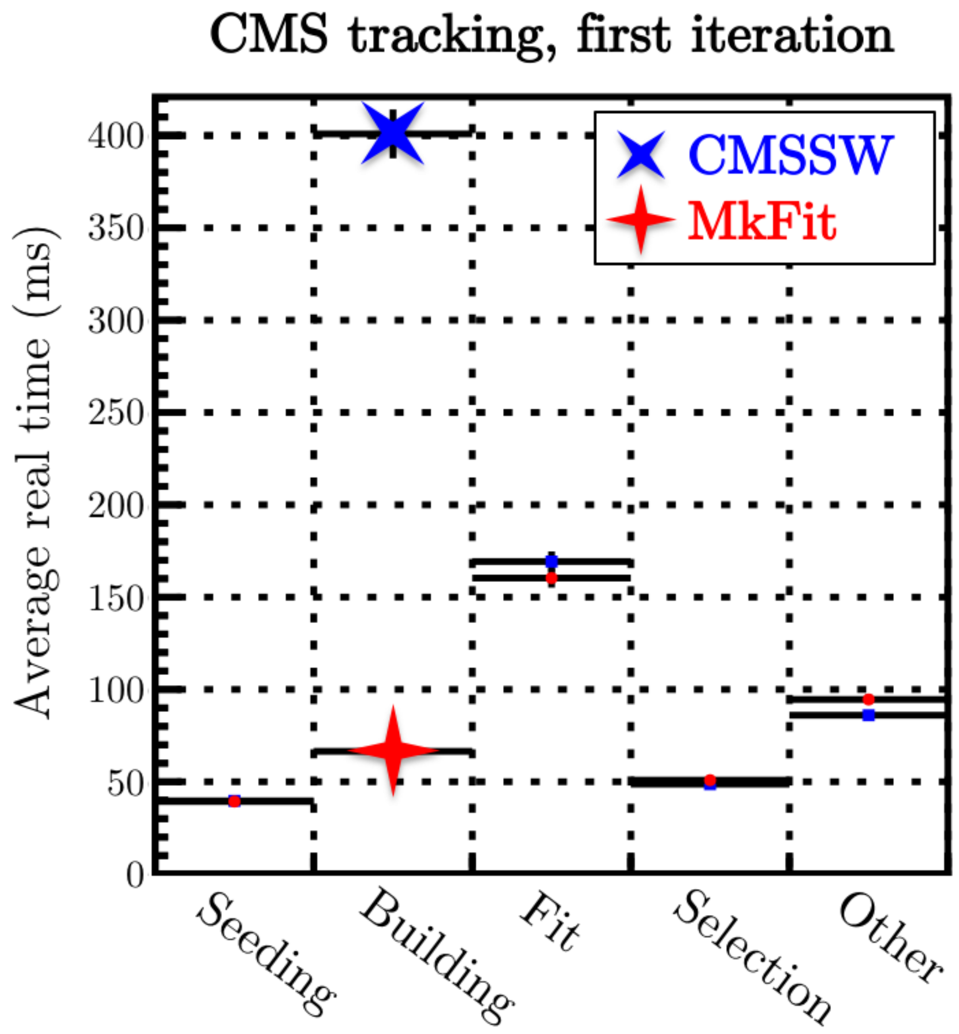


* Matriplex width set to 16, full processing time including I/O and setup

mkFit results: timing

- Can also run mkFit within CMSSW as an external package
- mkFit track building > **6x faster** than CMSSW, including all overheads
- Track building no longer dominates reconstruction time
- Results used a single thread: even larger speedups if multi-threaded

- **Ongoing work** to integrate with CMSSW for use in Run 3



* Measured on Intel SKL-SP, using simulated $t\bar{t}$ PU 50 events

* mkFit compiled with AVX-512, icc

* CMSSW compiled with SSE3, gcc

Outline

LHC and HL-LHC computing challenges → Reconstruction software: mkFit parallel tracking → Analysis software: Columnar analysis with Coffea

- CMS computing
- HL-LHC timeline
- Parallelization methods

- Track building
- mkFit parallel KF algorithm
- mkFit results

Columnar Object Framework For Effective Analysis (Coffea)

Documentation: https://coffeateam.github.io/coffea

Analysis workflow

Centrally produced data sets of recorded and simulated events

- Several tiers, each with reduced content
- RECO (Mb/ev) → AOD (500 kb/ev) → MiniAOD (50 kb/ev)

Ntupling (on grid)
Producing slimmed ROOT files with only the variables needed for your specific analysis

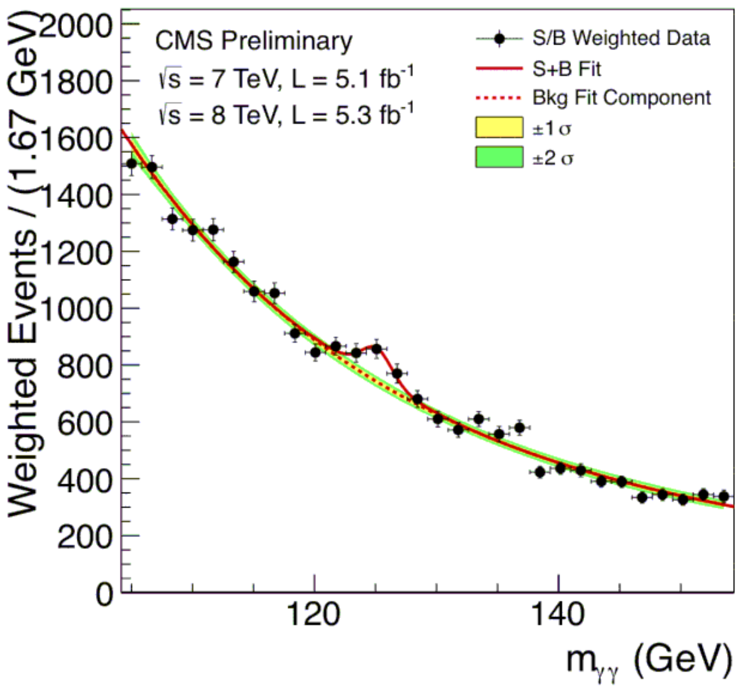
1–2 weeks, few times per year

Group ntuples or centrally produced **NanoAOD** (5 kb/ev)

Analysis code (locally or in batch)
Define signal and control regions, apply scale factors and corrections, estimate backgrounds, perform statistical analysis

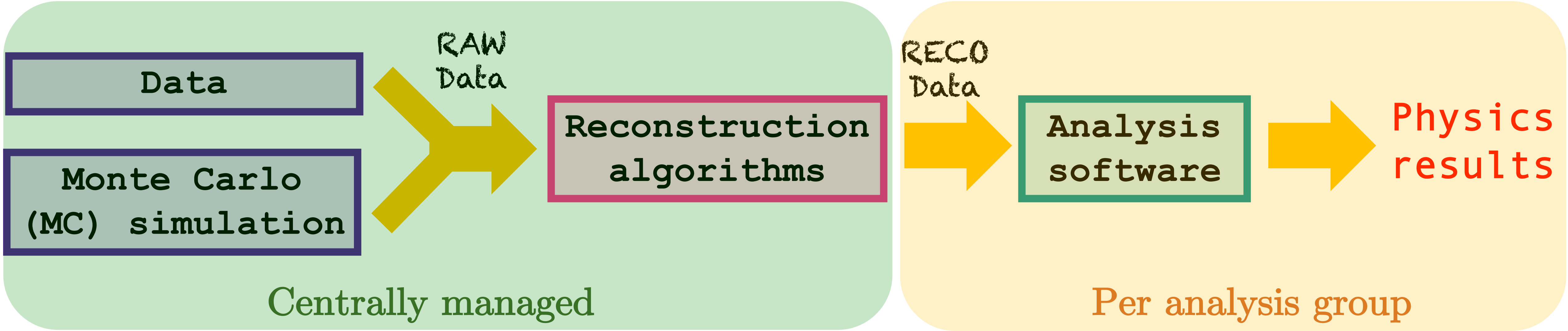
Several times per day

Final plots and tables



Analysis

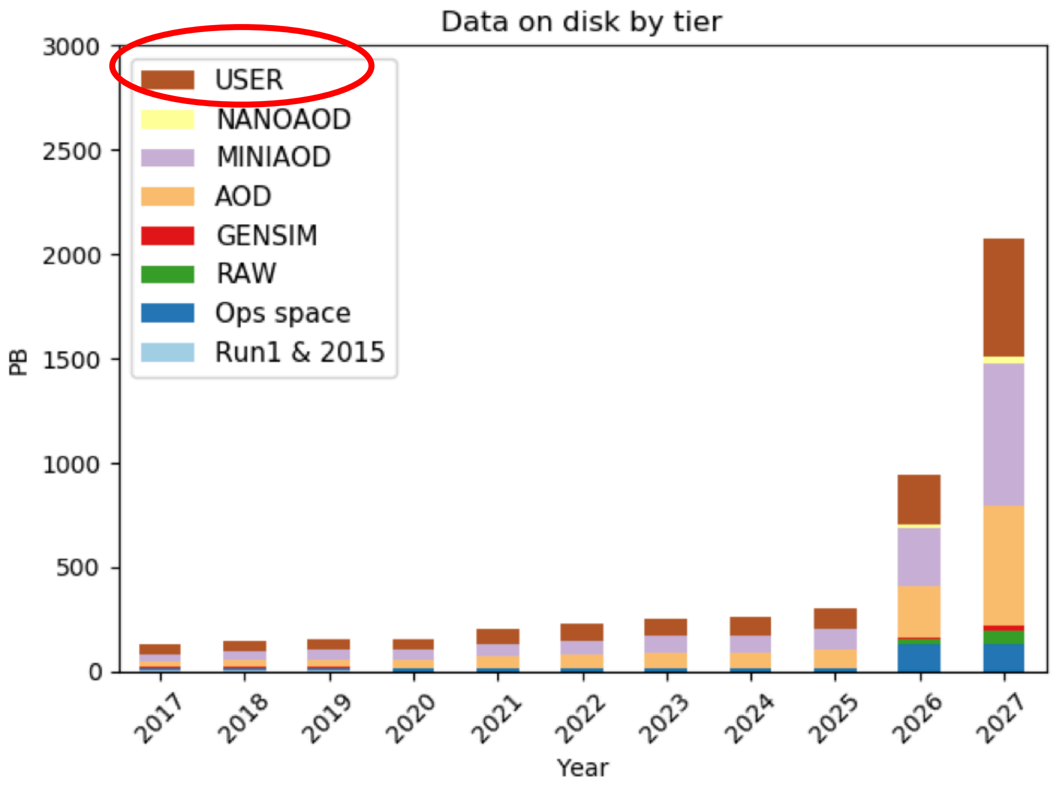
- Final step in CMS computing is the analysis software



- More than 100 analysis frameworks in CMS
  - Wide variation in efficiency, ease of use, computing languages
- Each group has their own ntuples
  - Inefficient use of CPU (to make the ntuples), disk resources (to store the ntuples), and personpower (for the manual bookkeeping of jobs, files, and datasets)
  - Partially solved by NanoAOD: centrally produced, containing all variables needed by approximately half of CMS analyses

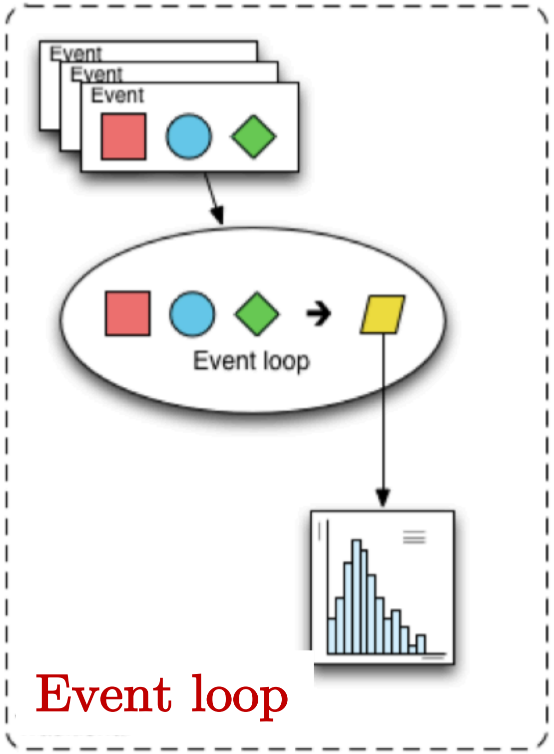
Motivation for Coffea

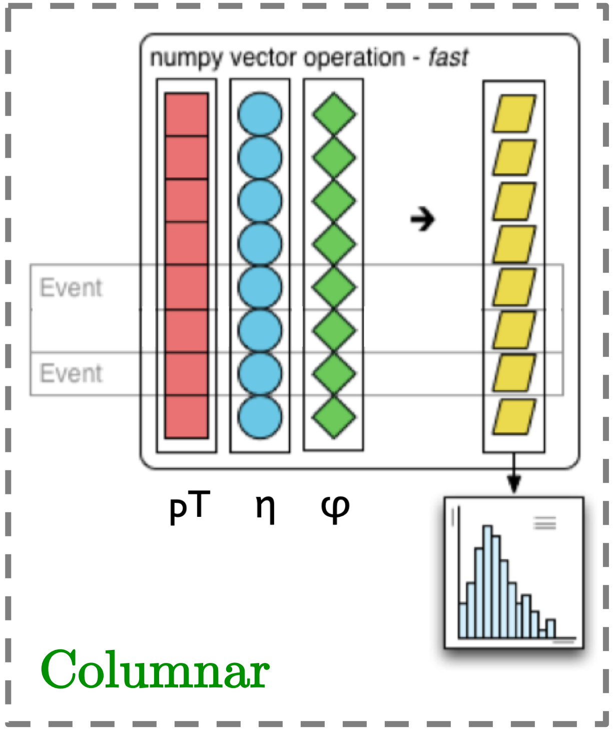
- Current disorganized analysis approach is not sustainable for the HL-LHC
- Need to minimize computing time, disk space, **and** physicist effort
- Coffea approach: move from ROOT-based tools to industry-standard techniques
  - Let the physicists worry about physics rather than manual setup and bookkeeping



Moving towards columnar analysis

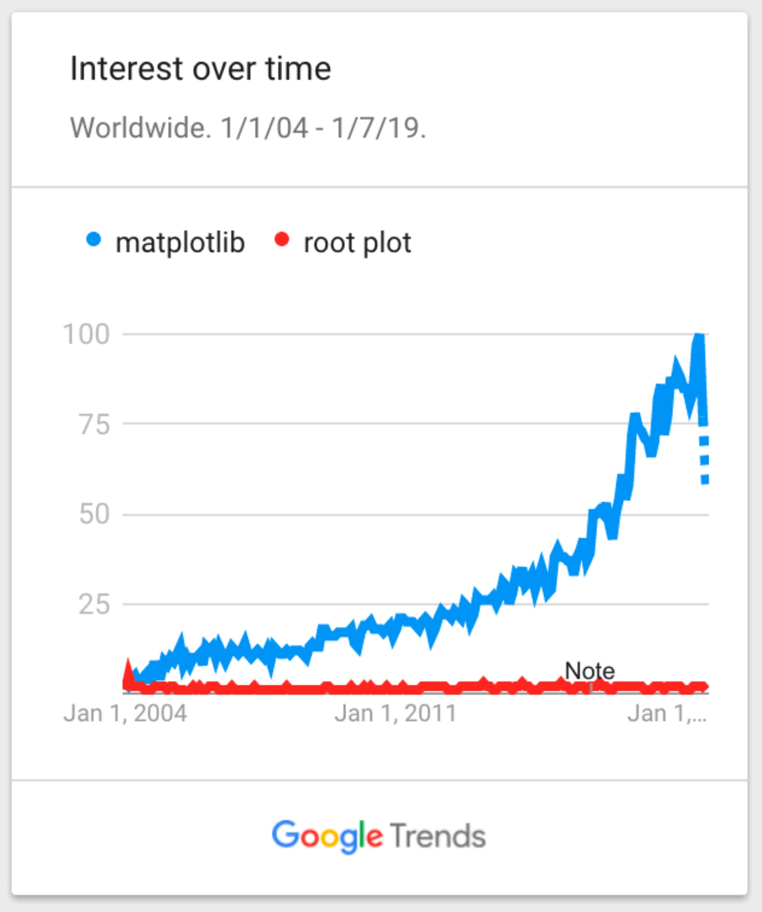
- Event loop analysis:
  - Load relevant values for a specific event
  - Evaluate several expressions
  - Store derived values
  - Repeat (explicit outer loop)
- Columnar analysis:
  - Load relevant values for many events into numpy arrays
  - Evaluate several **array programming** expressions
    - Operations which act on an entire array at once
    - Implicit *inner* loops
  - Store derived values
  - Utilizing scientific python: numpy, matplotlib





Strengths of columnar analysis

- Inherently vectorizable with efficient memory access
  - Takes much more effort to do the same using event loop analysis
- Array programming expressions in numpy are compiled C++
  - Much faster than a python *for* loop, avoids interpreter
- Minimizes disk space
  - Fast enough that there is no need to stage out intermediate steps
  - Work in progress: ability to add new columns to existing datasets in the database
    - For example, if you want to add displaced muons that are not stored by default in NanoAOD
- Ease of use: no need to write nested loops, filters by hand
- Easier to Google, prepares students for future careers
  - Students learn standard data science techniques



What is Coffea?

- Physicist-friendly tools for column-based analysis
- Implements typical recipes needed to operate on NanoAOD-like ntuples
- Uses scientific python ecosystem:
  - numpy, numba, scipy, matplotlib
  - Uproot: converts ROOT files into numpy arrays
  - Awkward-array: array programming primitives to handle "Jagged Arrays"

NumPy

uproot

matplotlib

Awkward Array

Muon pt: table

Event 1	40.2	25.6	10.2
Event 2	71.1	35.7	
Event 3	52.3		
Event 4	34.5	15.7	

Coffea framework

Back-end
Data delivery from ROOT ntuples into columns (awkward arrays)

Front-end
Control regions, systematics, corrections, histograms

Coffea provides:
- Support for several "column-delivery" mechanisms
- Choice of mechanism should be **transparent to the user**

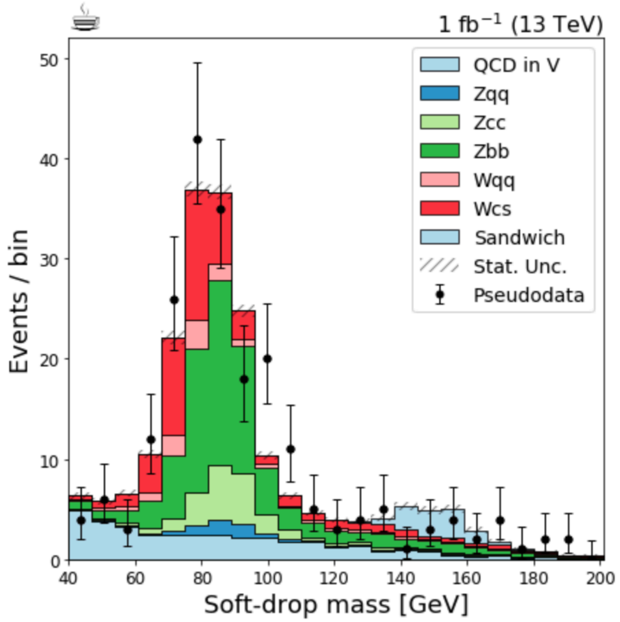
Coffea provides:
- Histogramming tools based on matplotlib
- Output to ROOT histograms if desired
- Lookup tools for weights and scale factors

uproot

DASK *

HTCondor High Throughput Computing *

APACHE Spark *



* Open-source software tools for distributed parallel computing

Analysis framework

- Coffea `processor` defines the analysis selections, weights, and output histograms (ie, the front-end analysis code)
  - Input: dataframe of awkward arrays
  - Output: histograms, counters, small arrays
- Coffea `executor` handles the interaction with the back-end scale out mechanism, such as communicating with HTCondor, a Spark cluster, or Dask
- Once defined, your `processor` can be passed to different `executors` with a single line change

Front-end code

- Idea of what it looks like in a real analysis
  - Python allows very flexible interface, under-the-hood data structure is columnar
- One line of code to define analysis objects and which columns you care about

```
eles = JaggedCandidateArray( events.nElectron,
                             ‘pt’ : events.Electron.pt,
                             ‘eta’: events.Electron.eta,
                             ‘id’ : events.Electron.cutBased)
```

- One line of code to select good electrons from all events - **no** explicit for loop over electrons!

```
clean_eles = eles[ (eles.pt > 7) & (abs(eles.eta) < 2.4) & ((eles.id&2) != 0) ]
```

- One line of code to define events passing signal region requirements - **no** explicit for loop over events!

```
selections['signal'] = pass_trigger & (clean_jets.counts == 1) & (met > 200) &
                       (clean_eles.counts==0) & (clean_muons.counts==0)
```

Backend: Coffea farms

- Dedicated **Analysis Facility** (AF) could provide the people, services, software, and hardware to run Coffea at scale with multiple users
  - Multiple facilities in development
  - Coffea Casa plans to invite alpha testers by the end of the year

Request data "over night" via data delivery service *(filter, add specific columns, optimise data layout for columnar analysis in AF)*

→

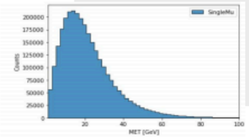
Request addition to existing NanoAOD *(request info from MiniAOD /correct existing branches)*

↓

Analyse custom NanoAOD in AF (as well in a batch from AF)

←

Get results on your laptop from AF



CMSAF @T2 Nebraska
"Coffea-casa"
https://cmsaf-jh.unl.edu

Elastic AF @ Fermilab

*Oksanna Shadura, IRIS-HEP workshop, Oct 27

Coffea status and plans

- Coffea is being used or explored for > 10 analyses in CMS
  - Including many people not on the development team
  - Also being used upstream by some of the Physics Object Groups to derive corrections and scale factors
  - Interest from other experiments such as DUNE
- Coffea is (relatively) easy to learn
  - Especially for those with no previous event-loop or ROOT experience
  - Code is easy to read, even for people used to C++ event loops
- Several Coffea farms under development
  - USCMS operations program is interested in supporting an analysis facility

- New collaborators/analyzers are welcome
  - Documentation: https://coffeateam.github.io/coffea

Outline

LHC and HL-LHC computing challenges

- CMS computing
- HL-LHC timeline
- Parallelization methods

Reconstruction software: mkFit parallel tracking

- Track building
- mkFit parallel KF algorithm
- mkFit results

Analysis software: Columnar analysis with Coffea

- Columnar analysis
- Coffea framework and front-end code
- Analysis facilities

Conclusions

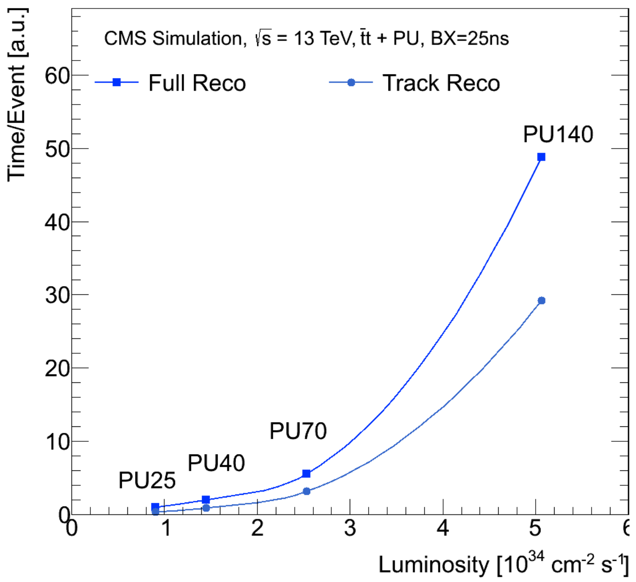
- CMS relies on sophisticated computing to achieve physics goals

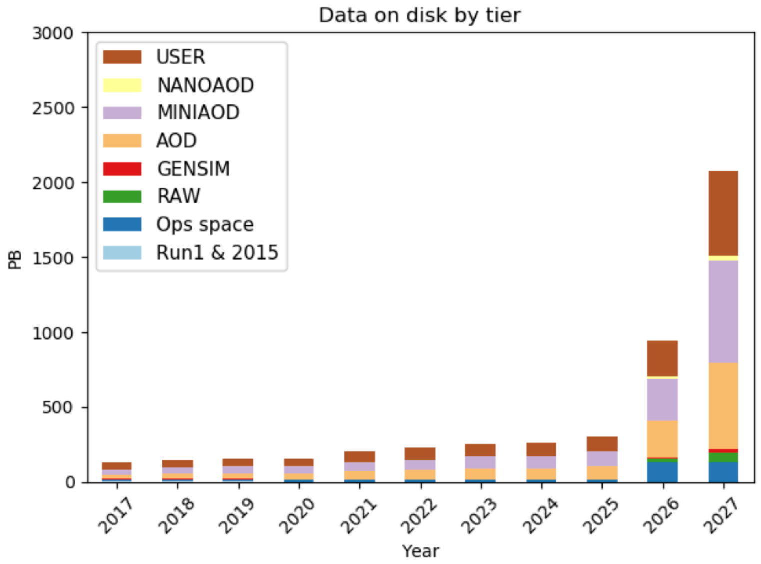
Data

Monte Carlo (MC) simulation

→ **Reconstruction algorithms** → **Analysis software** → **Physics results**

- HL-LHC presents significant computing challenges
- Bridge the gap through increased use of parallelization and optimized algorithms
  - mkFit speeds up track building by 6x
  - Coffea brings data science techniques to HEP data analysis





Thank you!

Application to other experiments

mkFit

- mkFit in theory could be used for ATLAS
  - Geometry is a plugin, factored out of main code
  - In practice, a lot of work comes down to interfacing with the peculiarities of the experiment's software framework
- Same optimization/parallelization approach can be applied to other experiments
  - Rewrote hit finding algorithm for LArTPC reconstruction; used in production for Icarus, 7x speedup
  - Exploring FFT algorithms
  - https://computing.fnal.gov/hepreco-scidac4

Coffea

- Working with ATLAS developers to expand its use
  - Not used by ATLAS analysis groups yet, mostly due to issues with file format
- Many neutrino experiments already use numpy + Pandas DataFrames for analysis
  - Coffea adds convenient histograms, lookup tools for uncertainties, ability to handle "awkward data"

Pokemon or Big Data?

- https://pixelastic.github.io/pokemonorbigdata/

Arvados

Big Data

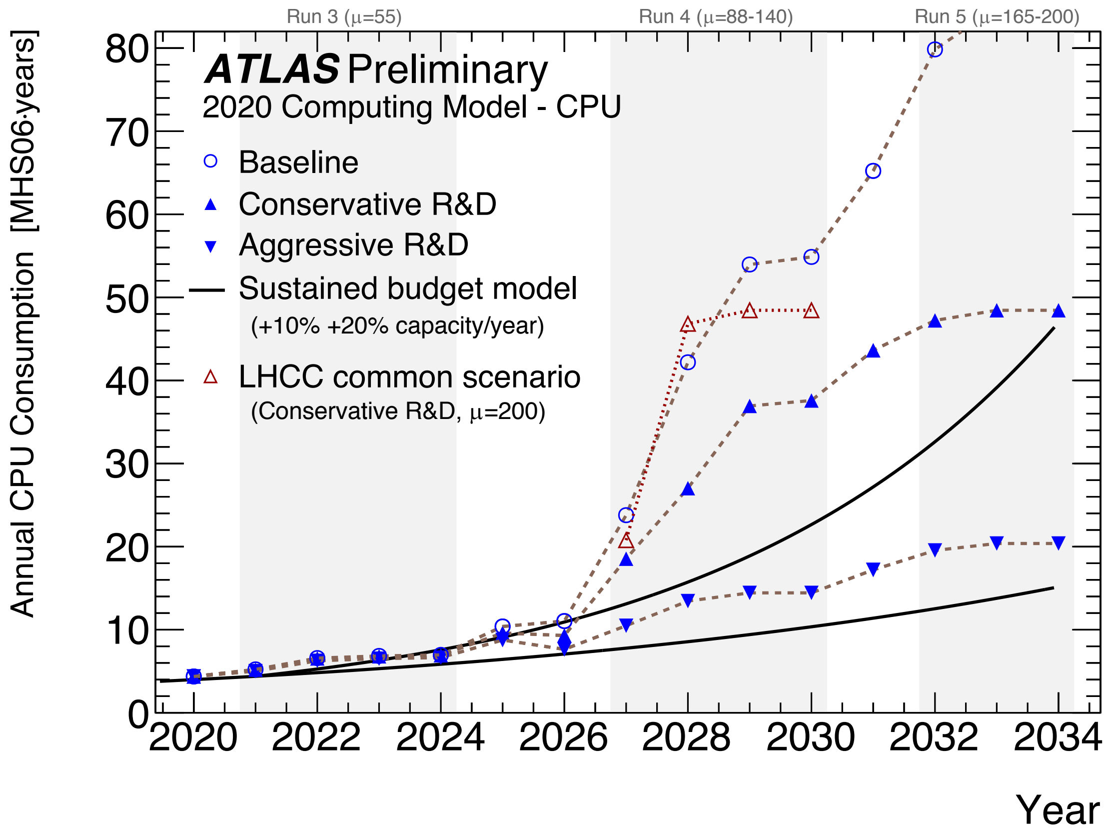
Pokemon

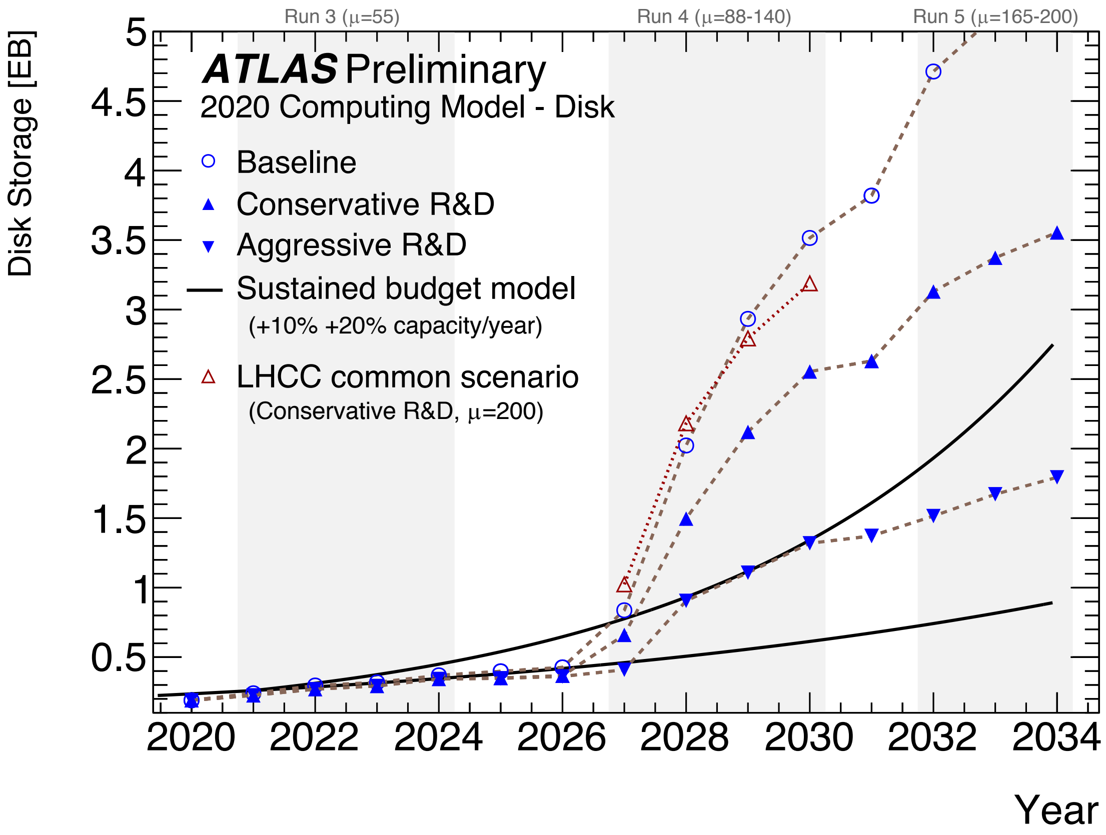
Arvados is Big Data!

Not to be confused with Ariados. Spins a web of microservices around unsuspecting sysadmins.

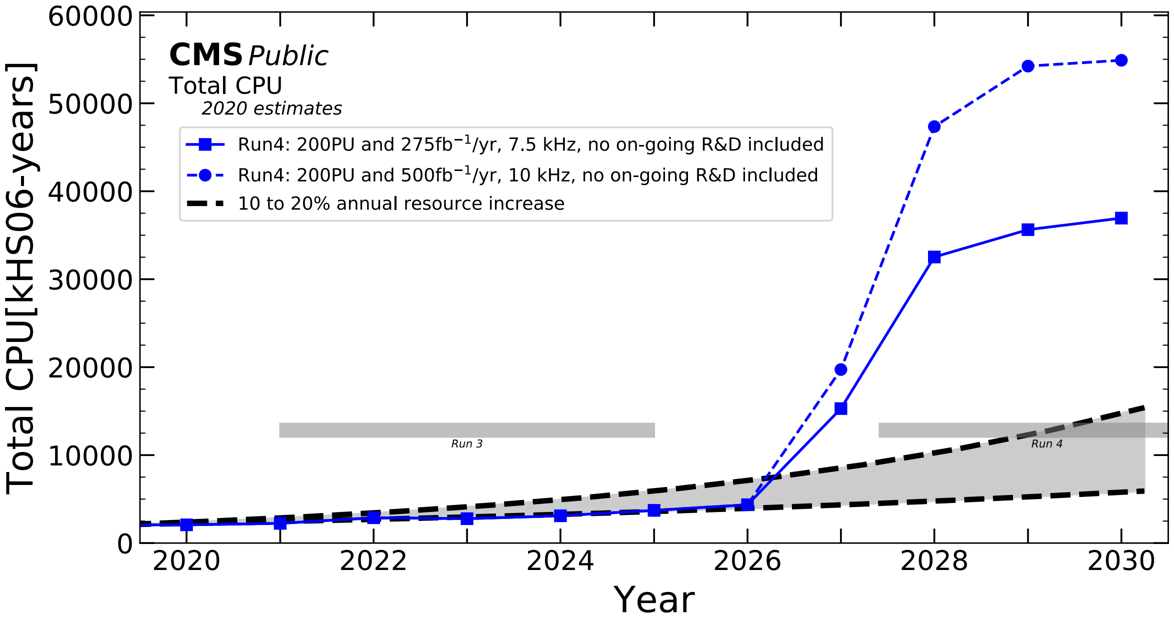
Next question

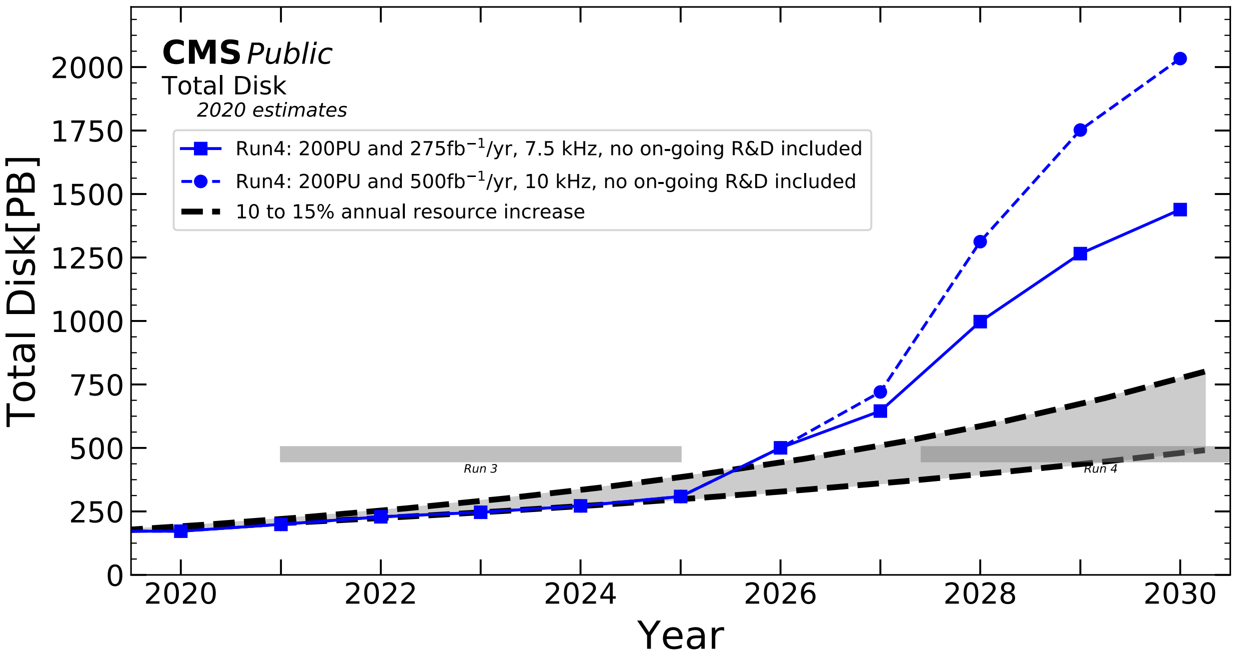
ATLAS computing during HL-LHC





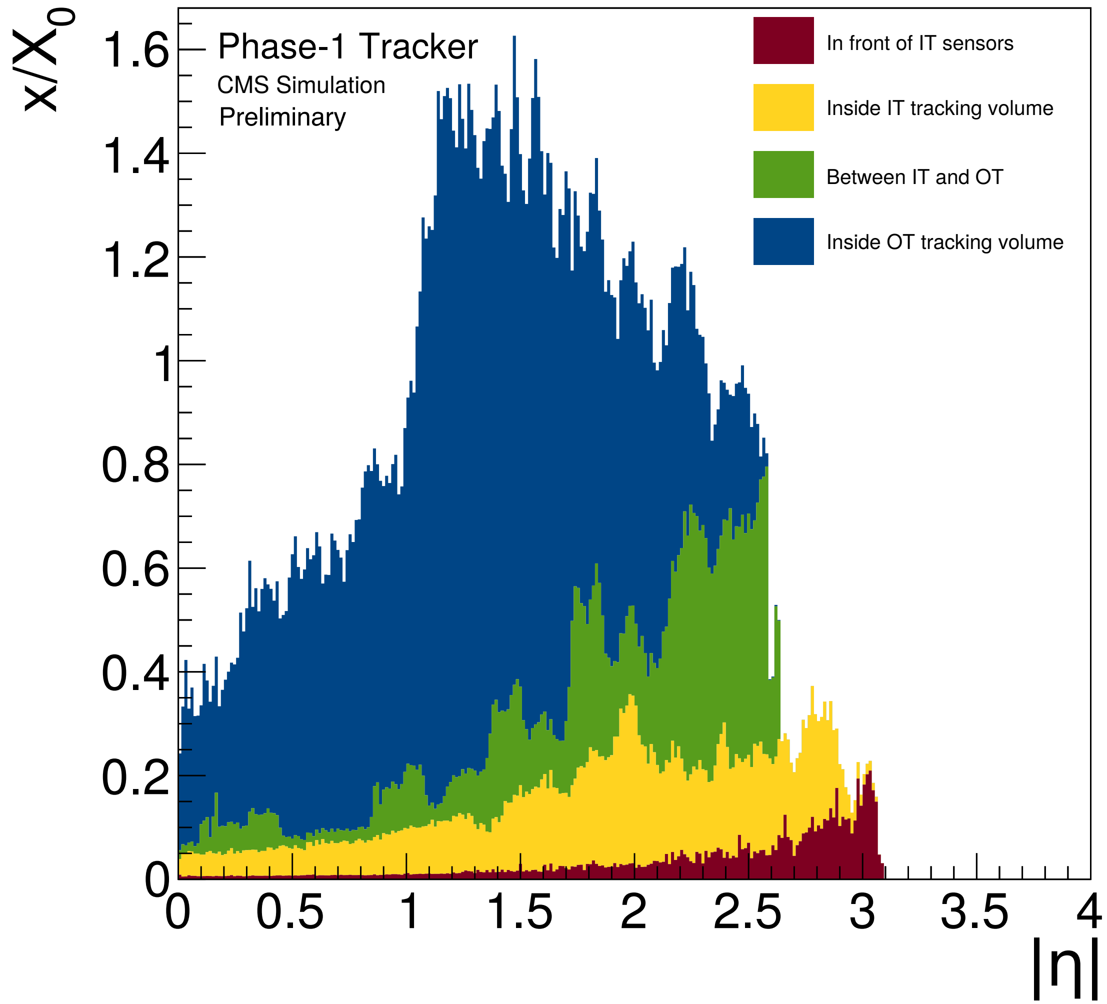
CMS computing during HL-LHC





Material, CMS tracker

- Amount of material in the CMS tracker is one of the primary reasons for using the KF algorithm for track building



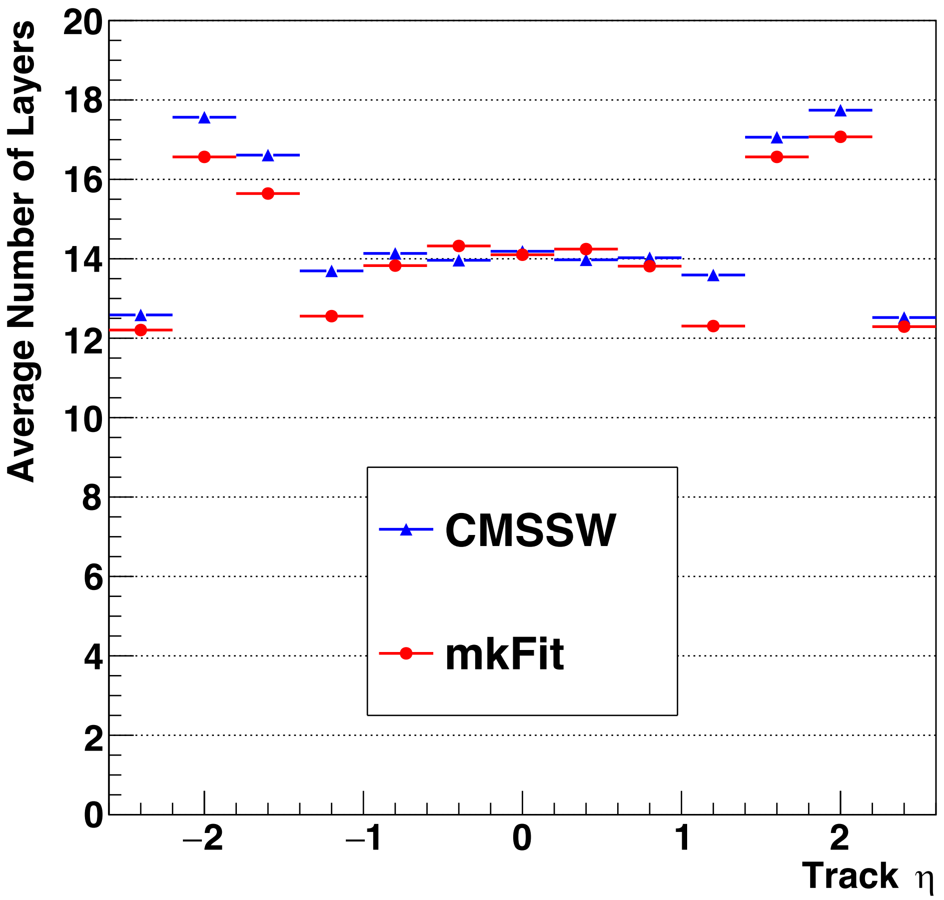
mkFit performance

- **Efficiency:** fraction of simulated tracks that are matched to a reconstructed track
- **Fake rate:** fraction of reconstructed tracks that are **not** matched to a sim. track
- **Duplicate rate:** fraction of sim tracks that are matched to $>$1 reco track

- **Simulated tracks** are required to be prompt, within acceptance ($|\eta| < 2.5$), and matched to a track seed
- Reconstructed tracks are considered **matched** to a simulated track if $> 75\%$ of hits are shared, including the seed
- Measured in standalone mkFit configuration (ie, not within CMSSW)
- Measured using simulated TTBar events with PU 50, realistic Phase I CMS geometry and detector conditions
- Computing performance tested on an Intel SKL-SP, dual socket x 16 cores

mkFit track quality

- mkFit finds hits on a comparable number of layers
- Caveats: showing number of layers rather than number of hits because mkFit originally could only pick up one hit /layer
  - In the case of overlapping modules, CMSSW can pick up both hits
  - Ongoing developments now to allow this to happen



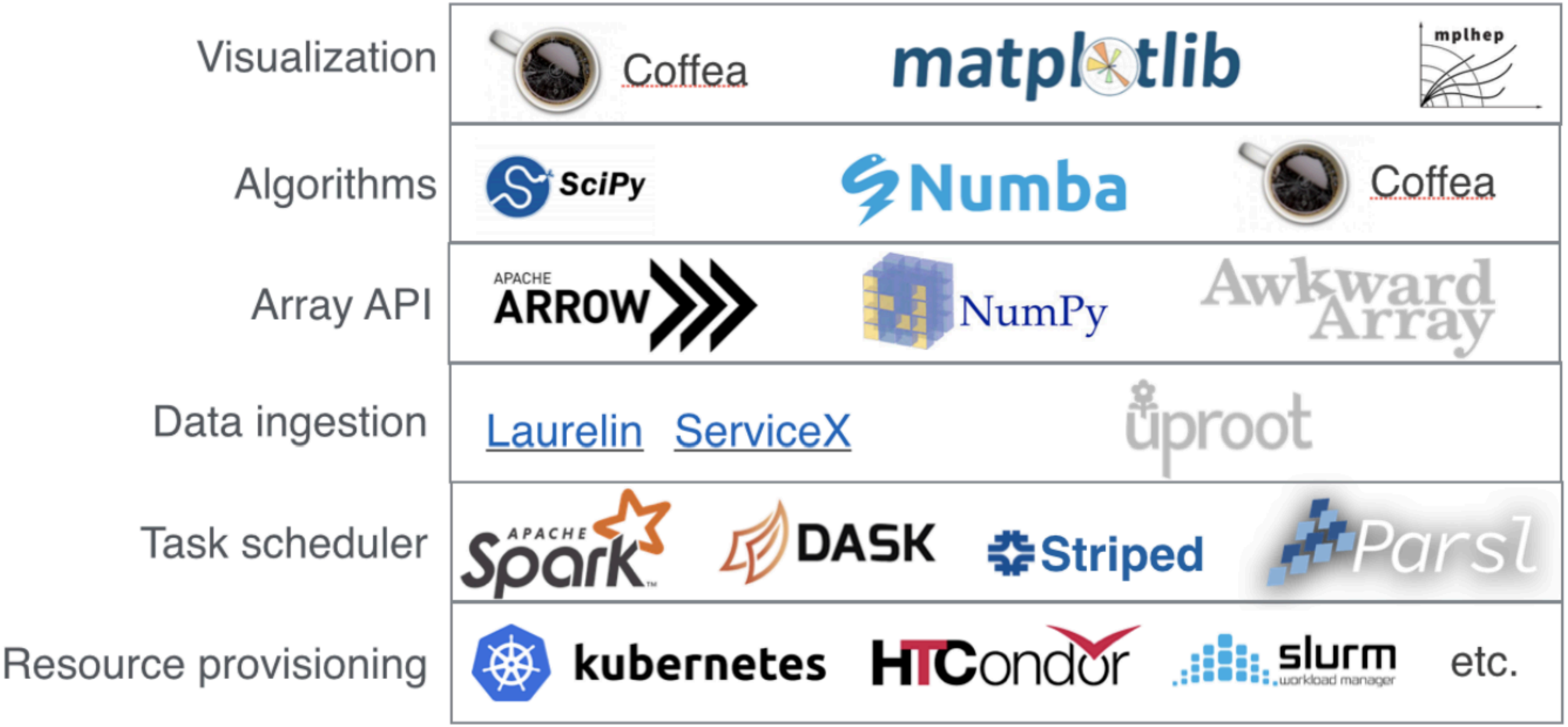
Coffea processor

- User is provided data frame of columns they wish to process
- User fills a defined set of accumulators
  - Histograms, dictionaries of counts, appendable arrays, ...
- Coffea executor takes care of the rest
  - Local machine, dask, spark, parsl (and condor)

```python
from coffea import hist, processor

class MyProcessor(processor.ProcessorABC):
    def __init__(self, flag=False):
        self._flag = flag
        self._accumulator = processor.dict_accumulator({
            # Define histograms
        })

    @property
    def accumulator(self):
        return self._accumulator

    def process(self, df):
        output = self.accumulator.identity()

        # PHYSICS GOES HERE

        return output

    def postprocess(self, accumulator):
        return accumulator

p = MyProcessor()
```

Coffea and scientific python

- Coffea fills in missing pieces of the software stack



Bigger picture of analysis

- Coffea spans much of the analysis workflow defined by the IRIS-HEP Analysis systems group